ENCICLOPEDIA DEL BASIC

LeG

SPECTRUM 5

Más allá del Basic



ceac

SPECTRUM 5
Más allá del Basic

ENCICLOPEDIA
DEL
BASIC

SPECTRUM 5

Más allá del Basic



Perú, 164 - 08020 Barcelona - España

Primera edición. Enero 1987

ISBN 84-329-7715-2 (Rústica)
ISBN 84-329-7720-9 (Rústica especial)
ISBN 84-329-7725-X (Cartoné)
ISBN 84-329-7732-2 (Cartoné especial)
ISBN 84-329-7726-8 (Cartoné. Obra completa)
ISBN 84-329-7727-6 (Cartoné. Obra completa especial)

Depósito Legal: B-44041 - 1986

Impreso por
GERSA. Industria Gráfica
Tambor del Bruc, 6
08970 Sant Joan Despí (Barcelona)

Printed in Spain
Impreso en España

Contenido

Aclaración importante

Observará el lector de esta «Enciclopedia del BASIC» que la numeración de capítulos es correlativa, iniciándose en el Tomo I de la misma y finalizando en el Tomo V y último.

Hemos creído conveniente tal continuidad, en primer lugar por el carácter secuencial de la Enciclopedia, al ser los temas que la constituyen correlativos entre sí y formando un todo inseparable; y en segundo lugar porque estimamos que ello facilita la *rápida localización de cualquier tema,* lo que, al tratarse de una obra de consulta frecuente, supone una indudable ventaja para el lector.

Cómo estudiar en esta Enciclopedia

Al comenzar cada Capítulo encontrará un *esquema de contenido*. La finalidad de este esquema es doble:

— Por una parte, le ofrece una visión panorámica de todos los temas que va a estudiar en ese capítulo.

— Por otra, si posteriormente debe repasar algún punto determinado, le facilitará el poder localizarlo.

Leálos despacio para tener esta visión panorámica.

Después del esquema de contenido, cada capítulo comienza definiendo sus objetivos. De esta manera usted sabrá desde el principio lo que aprenderá en él. También le servirá como referencia para saber si el objetivo marcado ha sido alcanzado por usted.

A lo largo de los capítulos y al final de los mismos encontrará unos *resúmenes*, seguidos de unos *ejercicios de autocomprobación*. Su finalidad es hacer un alto en el camino y recapitular lo estudiado desde la última parada. Al mismo tiempo, los ejercicios de autocomprobación le servirán para que usted mismo compruebe si ha asimilado los conceptos estudiados y si está en disposición de continuar adelante o, por el contrario, si es necesario volver a repasar algo antes de seguir. La solución a los ejercicios de autocomprobación la encontrará al final de la primera parte del volumen.

Le recomendamos también que, cuando deba interrumpir su estudio, lo haga siempre al final de un capítulo o al terminar de resolver alguno de los grupos de ejercicios de autocomprobación que aparecen a lo largo de las lecciones.

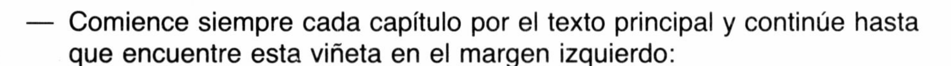
Al hablar de la composición de la Enciclopedia, le dijimos que cada volumen se componía de dos partes. La del texto principal o estudio del BASIC estándar, y la de «Práctica con el microordenador», que viene a ser un complemento de la anterior. Ahora que va a comenzar a estudiarlo comprenderá enseguida la razón de esta comparación.

Cuando se aprende un idioma es frecuente que uno se encuentre con la sorpresa de que en determinadas regiones aquello que uno aprendió a decirlo de una manera se diga de otra. Con el lenguaje de programación BASIC pasa lo mismo. Cada fabricante introduce en el uso de su ordenador un dialecto distinto, que normalmente coincidirá en su mayor parte con el

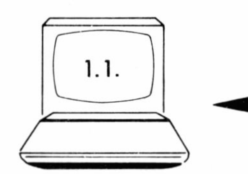
BASIC estándar, pero tendrá suficientes características especiales como para que el que comienza se encuentre ante su ordenador sin saber qué hacer en determinados momentos.

Por eso, como nuestra intención es que aprenda BASIC y que lo practique con *su microordenador* hemos utilizado la Enciclopedia de modo que usted no se encuentre perdido en ningún momento. Así, al estudiar la lección del BASIC estándar es como si dijéramos: *Esto se dice así normalmente en lenguaje BASIC*. Sin embargo, cuando haya diferencias, añadiremos en el capítulo «Prácticas con el microordenador»: *Ojo, que esto mismo en su microordenador se dice de esta forma*. De este modo, podrá dialogar tranquilamente con su microordenador sin desagradables sorpresas y traducir al lenguaje que entiende su microordenador un programa que en BASIC estándar pueda estar escrito de forma un poco diferente.

Concretamente, en el estudio de las dos partes debe proceder del modo siguiente:

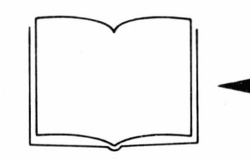


— Comience siempre cada capítulo por el texto principal y continúe hasta que encuentre esta viñeta en el margen izquierdo:



Verá que en la pantalla de la viñeta aparecen unos números. Concretamente 1.1. Esto indica que debe dejar en este momento el capítulo que está estudiando, y pasar al apartado 1.1. de «Prácticas con el microordenador».

— Continúe ahora el estudio de «Prácticas con el microordenador» hasta que encuentre esta otra viñeta:



En ella se le remite de nuevo al capítulo que dejó anteriormente, para continuar donde lo interrumpió. También aquí se le indica el sitio exacto donde debe continuar. En todo caso, le recomendamos que deje siempre una señal en la página donde interrumpe el estudio. Así tendrá siempre la página localizada.

Observará el lector de esta «Enciclopedia del BASIC» que la numeración de capítulos es correlativa, iniciándose en el Tomo I de la misma y finalizando en el Tomo V y último.

Hemos creído conveniente tal continuidad, en primer lugar por el carácter secuencial de la Enciclopedia, al ser los temas que la constituyen correlativos entre sí y formando un todo inseparable; y en segundo lugar porque estimamos que ello facilita la *rápida localización de cualquier tema*, lo que, al tratarse de una obra de consulta frecuente, supone una indudable ventaja para el lector.

Parte I
BASIC

Capítulo 16
• Estudio de algoritmos

ESQUEMA DE CONTENIDO

El concepto de algoritmo.

Procesos de clasificación.	Algoritmo de selección. Algoritmo de burbuja. Algoritmo de Shell.
Procesos de búsqueda.	Algoritmo de búsqueda secuencial. Algoritmo de búsqueda binaria.

Intercalación o fusión.

Otro ejemplo práctico: el calendario.

16.0 OBJETIVOS

El objetivo de este capítulo es la presentación de un concepto que, aunque no forma parte del lenguaje BASIC, está presente en todos los programas. Este es el concepto de algoritmo.

De una forma simplificada, podemos decir que un algoritmo es un procedimiento para resolver un problema.

El lenguaje no hace más que expresar este procedimiento, de forma que resulte inteligible para el ordenador. Por tanto, si el procedimiento es bueno y lo traducimos correctamente, obtendremos un programa adecuado para resolver el problema.

Si por el contrario, el procedimiento no es bueno (en determinados casos resulta ambiguo, da errores, presenta fallos en su funcionamiento, etc.), aunque hagamos la traducción correcta, el programa que obtendremos será erróneo; no nos servirá para nuestro objetivo, que es la solución del programa.

En este capítulo pues, aprenderemos el concepto de algoritmo, y veremos además algunos ejemplos de su utilización en la resolución de problemas que suelen presentarse frecuentemente.

Así, veremos diversos procedimientos para ordenar listas (algoritmo de selección, algoritmo de burbuja, algoritmo de Shell), algunos procedimientos para buscar un elemento en una lista (algoritmo de búsqueda secuencial y algoritmo de búsqueda binaria), un procedimiento de mezclar listas (algoritmo de intercalación o fusión), así como un ejemplo práctico que incluye varios algoritmos, consistente en la realización de un calendario.

Estos algoritmos le resultarán útiles no sólo en cuanto al aprendizaje de los procedimientos en sí mismos, sino también en cuanto a la forma en que los programas se han escrito, intentando no sólo facilitar su lectura y comprensión, sino también conseguir la máxima eficacia.

16.1 EL CONCEPTO DE ALGORITMO

El presente capítulo se va a dedicar al estudio de algunos problemas que surgen muy frecuentemente en el campo de la programación.

Estos problemas, que quizá se le presenten por primera vez, se han planteado ya desde hace muchos años y a muchas personas antes que a usted. Evidentemente, puede tratar de resolverlos por su cuenta e, incluso, es aconsejable que dedique un tiempo a tratar de buscar la solución. Sin embargo, dada la importancia de los problemas que se pueden plantear, ya sea por la frecuencia con que se encuentran, o por la dificultad de los mismos, vamos a ver algunos procedimientos generales, que nos permitirán obtener la solución más adecuada en cada caso.

Estos procedimientos reciben el nombre de algoritmos.

Algoritmo

Un algoritmo puede definirse pues, como *una secuencia ordenada de pasos y de decisiones concretas, exenta de ambigüedades, que lleva a la solución de un problema determinado.*

Debe tener en cuenta que un problema idéntico puede presentarse bajo apariencias muy distintas. Por esta razón, el programador principian-

te tiene tendencia generalmente a tratar de resolverlo para el caso concreto que se le plantea, sin ver la estructura general, tanto del problema como de la supuesta solución al mismo. En cuanto se modifique ligeramente el problema, se verá forzado a plantearse de nuevo la resolución, y esto deberá hacerlo tantas veces como cambien los datos. De esta forma, la falta de un enfoque global, hará que tenga que efectuar repetidamente el mismo trabajo, sin que por otra parte ofrezcan demasiadas garantías las supuestas soluciones que adopte en cada caso.

Para ilustrar de alguna forma lo que estamos diciendo, trate de plantearse el problema de ordenar los datos de una lista que puede suponer que son números.

Ordenación de una lista

Empecemos considerando una lista de dos elementos, que deseamos ordenar de menor a mayor. El programa para ello podría ser:

```
10 INPUT A
20 INPUT B
30 PRINT "LA LISTA A ORDENAR ES: ";A,B
40 IF A<B THEN GOTO 70
50 PRINT "LA LISTA ORDENADA ES: ";B,A
60 GOTO 80
70 PRINT "LA LISTA ORDENADA ES: ";A,B
80 END
```

El programa empezará preguntando los dos valores que se deben ordenar. Una vez los hayamos entrado, los escribirá (en el mismo orden en que se le hayan introducido).

Si el primer elemento (contenido en la variable *A*) es menor que el segundo (contenido en *B*), los podremos escribir en el mismo orden en que han sido introducidos. En caso contrario –si el elemento *A* es mayor que el elemento *B*– deberemos escribirlos poniendo en primer lugar el valor *B* y en segundo lugar el valor *A*, tal como se indica en el programa.

Ejecutando el programa, observará que nos escribe los números en el orden deseado. Pero en realidad lo que hemos hecho no ha sido estrictamente ordenar los números, sino que únicamente los hemos escrito en el orden adecuado. La variable *A* sigue conteniendo un número mayor que la variable *B*, tal como los hemos introducido.

Para hacer efectiva la ordenación, deberíamos proceder de forma distinta: si el contenido de *A* es ya inferior al de *B*, no hay que efectuar modificación, si por el contrario el valor de *A* es mayor que el de *B*, habrá que intercambiar sus valores, pasando el valor de *A* a la variable *B*, y el valor de *B* a la variable *A*. El programa para ello será:

```
10  INPUT A
20  INPUT B
30  PRINT "LA LISTA A ORDENAR ES: ";A,B
40  IF A<B THEN GOTO 80
50   LET T=A
```

```
60   LET A=B
70   LET B=T
80  PRINT "LA LISTA ORDENADA ES: ";A,B
90 END
```

Observe en este caso la forma de realizar la ordenación. Si los números se introducen de forma que el valor de *A* es menor que el de *B*, el programa procede a escribirlos directamente. Por el contrario, si el valor del primero es superior al del segundo, debemos intercambiarlos. Para ello, necesitaremos una variable auxiliar, tal como hemos hecho en el programa, utilizando la variable *T*. Quizás a primera vista no parezca necesario incluir esta variable, dado que si lo que queremos es hacer *A* igual a *B*, y *B* igual a *A*, bastaría hacer:

```
LET A=B
LET B=A
```

¿Por qué no lo podemos hacer directamente de esta forma?

En la figura 1 se representa gráficamente lo que pasa al hacer esta doble asignación. Al igual *A* a *B*, el valor que contenía la variable *A* (1) es sustituido por el de *B* (3), con lo que perdemos el valor de *A*. Al hacer ahora *B* = *A*, lo único que conseguimos es asignar de nuevo a *B* el valor que ya tenía (3). Por tanto, es necesario almacenar el contenido de *A*

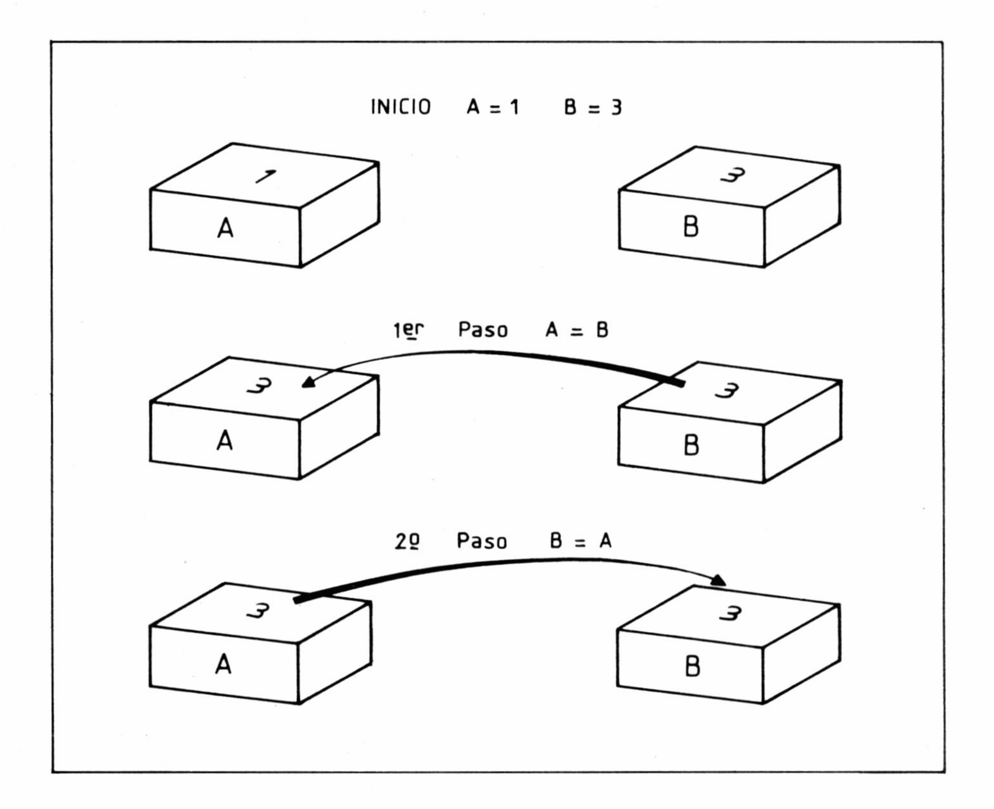


Figura 1: Asignación de variables.

antes de hacer la asignación $A = B$ pues de lo contrario perderíamos este valor.

El proceso a seguir es pues el que se muestra en la figura 2. Vemos que en este caso hemos hecho efectivo el intercambio de los valores de A con B, tal como deseábamos.

Resolviendo el problema de esta forma, hemos conseguido ordenar dos números. Vamos a plantearnos ahora el siguiente paso, que consistirá en ordenar tres números. Podríamos intentar resolver el problema para este caso particular, utilizando tres variables para contener los tres números a ordenar, más una variable auxiliar para los intercambios. Sin embargo, si lo planteamos de esta forma, cuando queramos seguir avanzando y ordenar cuatro números, deberemos plantearnos el problema de nuevo, y resolverlo para este nuevo caso particular, y así, buscar la solución particular para cada caso, según el número de elementos de la lista a ordenar.

Generalización del problema de ordenar listas

Evidentemente esto no resulta práctico. Es necesario encontrar una manera general de resolver este problema, de forma que nos permita ordenar una lista de cualquier tamaño. Dado que, tal como ya hemos dicho, este problema se plantea muy a menudo, se han desarrollado diversos algoritmos de ordenación, de tipo general. En el siguiente apartado veremos algunos de estos algoritmos.

16.2 PROCESOS DE CLASIFICACIÓN

Vamos a plantearnos el problema de ordenar una lista cualquiera. Para empezar, supondremos que se trata de una lista de números, que queremos ordenar en orden decreciente; es decir, de mayor a menor.

Si la lista es muy corta, la ordenación podemos hacerla inmediatamente. Así, si tenemos la lista:

7, 5, 94, 0

después de leerla, podremos ordenarla sin ni siquiera escribirla; la ordenación la hemos hecho mentalmente. El resultado será:

94, 7, 5, 0

Supongamos ahora que se trata de ordenar una lista más larga; por ejemplo una lista de 20 elementos:

3, 5, 44, 0, 92, 7, 63, 8, 6, 102, 6, 23, 1, 4, 12, 61, 54, 98, 20, 33

Si tenemos muy buena memoria, tal vez podamos ordenarla y recordarla sin necesidad de escribir nada. Pero, para asegurar que no nos olvidamos de ningún elemento, será mejor que vayamos escribiendo una nueva lista, con los elementos ya ordenados. Cada elemento que pasa a formar parte de esta segunda lista se tacha de la lista inicial.

Procedimiento mediante dos listas: la inicial y la ordenada

En este caso nos ayudamos de papel y lápiz, pero el proceso que hemos seguido mentalmente es el mismo, tanto si la lista es larga como

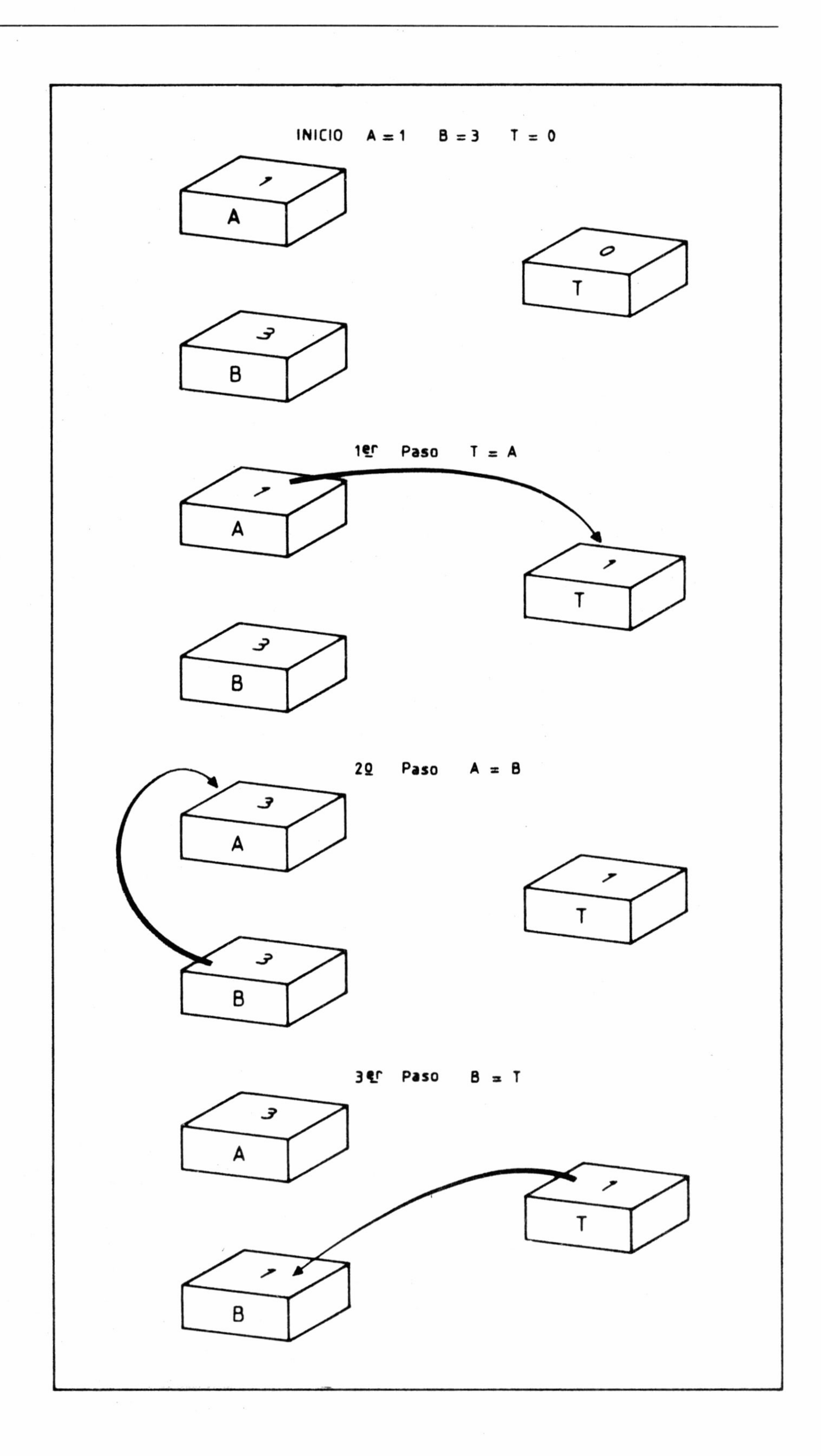


Figura 2. Intercambio de variables.

si es corta. En primer lugar, buscamos entre todos los elementos de la lista inicial cuál es el mayor, lo tachamos y lo escribimos en la nueva lista. Veámoslo para la primera lista, más corta. Tendremos

Lista inicial:	7, 5, ~~94~~, 0
Lista ordenada:	94

Seguiremos el proceso buscando de nuevo entre los elementos que quedan en la lista inicial, tachamos el mayor y lo escribimos en la lista final. Tendremos ahora:

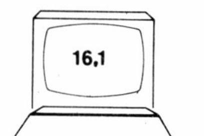
Lista inicial:	~~7~~, 5, ~~94~~, 0
Lista ordenada:	94, 7

Este proceso se deberá repetir hasta tachar todos los elementos de la lista inicial, con lo que tendremos la lista ordenada completa:

Lista inicial:	~~7~~, ~~5~~, ~~94~~, ~~0~~
Lista final:	94, 7, 5, 0

De esta forma, hemos ordenado la lista tal como deseábamos.

El procedimiento seguido no es el único posible, existen otras formas de realizar la ordenación. En la siguiente sección veremos cómo escribir el programa correspondiente al procedimiento que hemos seguido en este caso, y en secciones sucesivas, otros algoritmos de claficiación.



16.2.1 Algoritmo de selección

El algoritmo de selección sigue el proceso que hemos descrito. En su forma más simple, utiliza dos listas, la lista inicial, que contiene los elementos desordenados, y una lista final, en la que iremos colocando los valores ordenados. Para «tacharlos» de la primera lista, lo que se hace es substituirlos por un valor especial, que no interfiera con los elementos de la lista (por ejemplo el 9999).

Intercambio del elemento mayor con el primero de la lista

Sin embargo, existe una forma mejorada del algoritmo, que evita tener que utilizar dos listas y tener que ir «tachando» –es decir substituyendo– los elementos que hayamos ordenado. Para ello, lo que se hace es trabajar sobre una sola lista. Sobre la lista inicial, que contiene los elementos desordenados, se busca el elemento mayor, tal como hacíamos antes. Pero en lugar de sacarlo de la lista, lo que se hace es intercambiarlo con el elemento que ocupa la primera posición. A continuación se busca el siguiente número mayor, a partir de la segunda posición de la lista, y se intercambia con èl que ocupa este lugar. A continuación se buscará el tercero, y se intercambiará, después el cuarto, y así sucesivamente hasta llegar al final de la lista.

Pasos en la ordenación

Así, si deseamos ordenar de mayor a menor la lista:

7, 5, 94, 0

el proceso a seguir será el siguiente:

Paso 1: Buscar el elemento mayor de la lista, empezando por la primera posición (posición 1).

Lista actual → 7, 5, 94, 0

Elemento mayor → 94

Paso 2: Intercambiar este número con el que ocupa la posición 1. Tendremos pues:

Lista actual → 94, 5, 7, 0

Paso 3: Bucar el elemento mayor a partir de la segunda posición (posición 2). Tendremos:

Elemento mayor → 7

Paso 4: Intercambiar este valor con el que ocupa la segunda posición. Entonces:

Lista actual → 94, 7, 5, 0

Paso 5: Buscar el elemento mayor a partir de la tercera posición (posición 3). Dado que sólo queda un número –el 0– éste será el elemento que compararemos con el que ocupa la posición 3.

Elemento mayor → 5

Paso 6: Intercambiar –si es necesario– este valor con el que ocupa la tercera posición. En este caso no es necesario, con lo que tendremos:

Lista actual → 94, 7, 5, 0

El proceso ha terminado, ya que hemos llegado al final de la lista, por tanto, podemos asegurar que la lista está correctamente ordenada.

Si observamos la lista, vemos que a partir del Paso 4 ya la teníamos ordenada. Sin embargo, no debemos olvidar que nosotros tenemos una ventaja sobre el ordenador: vemos la lista completa, distinguimos directamente si está o no ordenada. Evidentemente la máquina no posee esta visión global del los números, por lo que no le queda más remedio que ir comparando sucesivamente todos los valores hasta el final.

Por otra parte, considerando la forma de trabajar de la máquina, en los pasos impares (Paso 1, Paso 3, Paso 5) en que buscamos el mayor entre varios elementos, vemos que no lo puede determinar directamente. Así, estos pasos deberían desglosarse en operaciones más simples, dado que la máquina sólo puede determinar cuál es el mayor valor de la lista si compara uno a uno todos los elementos entre sí. El Paso 1 en forma detallada sería:

Paso 1.1: Comparar el elemento en posición 1 con el elemento en posición 2. Tomar el mayor entre ellos.

elemento mayor → 7

Paso 1.2: Comparar el elemento mayor con el que ocupa la siguiente posición (la 3). Tomar el mayor entre ellos:

Elemento mayor → 94

Paso 1.3: Comparar el elemento mayor con el que ocupa la posición 4, tomando el mayor entre ellos.

Elemento mayor → 94

Dado que el elemento cuarto es el último de la lista, podemos asegurar que tenemos el mayor. De la misma forma deberíamos ir desglosando los pasos 3 y 5. El procedimiento en ellos es el mismo, únicamente varía la posición a partir de la cual se comparan los números.

Viendo el procedimiento con detalle, observamos que todo el rato realizamos únicamente dos operaciones:

- Comparar dos números.
- Intercambiar dos números.

Estas operaciones se realizan sucesivamente sobre todos los elementos de la tabla, ya que en realidad vamos comparando de izquierda a derecha cada número con todos los situados más a su derecha, intercambiándolos si es necesario para situar el mayor en posición correcta.

Una variante del algoritmo consiste en intercambiar los elementos sucesivamente al irlos comparando. Es decir, en lugar de tomar un elemento, buscar entre todos los de su derecha cuál es el mayor y si es necesario intercambiarlos, lo que hacemos es ir comparando el elemento en una posición determinada con cada uno de los de su derecha, y realizando directamente el intercambio cuando se encuentra uno mayor. Siguiendo este procedimiento, el listado del program para ordenar de mayor a menor una lista de números es el siguiente:

Algoritmo de selección

Programa del «Algoritmo de selección»

```
 NEW
 10 REM ==============================================
 15 REM ORDENACION; Algoritmo de seleccion
 20 REM ----------------------------------------------
 30 CLS
 35 REM Introduccion de la lista de numeros
 40    INPUT "Numero de elementos de la lista: ";N
 50    DIM A(N)
 55    FOR I=1 TO N
 57          PRINT "Elemento ";I;
 60          INPUT A(I)
 70          NEXT I
 80 REM Procedimiento de ordenacion
 90    FOR I=1 TO N-1
100          FOR J=I+1 TO N
110            IF A(I)>A(J) THEN GOTO 150
120            LET T=A(I)
130            LET A(I)=A(J)
```

```
140          LET A(J)=T
150          NEXT J
160         NEXT I
170 REM Impresion del resultado
175    CLS : PRINT "LISTA ORDENADA" : PRINT
180    FOR I=1 TO N
190         PRINT A(I);" ";
200         NEXT I
210 END
```

Veamos el funcionamiento del programa:

En primer lugar –línea 40– introducimos el número de elementos de la lista, es decir, la cantidad de números a ordenar.

A continuación –líneas 55 a 70– introduciremos estos elementos, después de dimensionar convenientemente la variable que ha de contener la lista (línea 50). Recuerde que las variables *I* y *J* contienen la posición de los elementos; es decir, la posición primera, segunda, tercera, etc., y las variables *A (I)* y *A(J)* el valor del elemento de la respectiva posición. *T* es una variable auxiliar que nos permite realizar el intercambio de los elementos sin perder su valor.

El procedimiento de ordenación se realiza entre las líneas 90 y 160. Este procedimiento está constituido por dos bucles: Uno más exterior, que va recorriendo uno a uno los elementos de la lista; y otro más interior, que utilizamos para ir comparando cada elemento con los que están más a su derecha (observe que este bucle se mueve desde la posición *I+1*, siendo *I* la posición del elemento a considerar, hasta el final de la lista). De esta forma comparamos el elemento en posición *I*, con los elementos en posición *J*, que tal como hemos dicho son los que están a su derecha. Cuando encontramos un valor mayor, lo intercambiamos (líneas 120 a 140), y seguimos recorriendo la lista. Por último, en las líneas 180 a 200 se escribe la lista resultante de la ordenación.

Veamos cómo se desarrollará la ordenación, sobre la lista que antes considerábamos. Los resultados los expondremos con la tabla de valores de las variables.

Lista de entrada: 7, 5, 94, 0

En la tabla de variables (Fig. 3) vemos más claramente cómo funciona el algoritmo. Para cada valor de la variable *I*, la variable *J* se mueve desde *I+1* hasta el final de la lista. En las correspondientes columnas se muestran los valores que se van comparando sucesivamente (*A(I)* con *A(J)*), y en la última columna se muestra la lista después de realizar la comparación, y si es necesario, el intercambio.

16.2.2 Algoritmo de burbuja

Vamos a ver ahora otro algoritmo de ordenación, denominado algoritmo de burbuja. El algoritmo recibe este nombre debido a la forma en

Figura 3. Tabla de variables en el algoritmo de selección.

					Efecto del IF			
PASO	**I**	**J**	**A (I)**	**A (J)**	**T**	**A (I)**	**A (J)**	**Resultado**
1	1	2	7	5	–	7	5	7,5,94,0
2	1	3	7	94	7	94	7	94,5,7,0
3	1	4	94	0	–	94	0	94,5,7,0
4	2	3	5	7	5	7	5	94,7,5,0
5	2	4	7	0	–	7	0	94,7,5,0
6	3	4	5	0	–	5	0	94,7,5,0

que evoluciona la lista que se ordena. Cada elemento se compara con su vecino, y se intercambia si es necesario. Esto hace que los elementos más pequeños vayan subiendo hacia el final de la lista paso a paso, de forma semejante a una burbuja que avanza hacia la superficie del agua.

El programa para realizar el algoritmo de burbuja es el siguiente:

Algoritmo de burbuja

Programa del «Algoritmo de burbuja»

```
NEW
 10 REM ========================================
 20 REM ORDENACION; Algoritmo de burbuja
 30 REM ----------------------------------------
 40 CLS
 50 REM Introduccion de los elementos
 60   INPUT "Numero de elementos de la lista: ";N
 65   DIM A(N)
 70   FOR I=1 TO N
        PRINT "Elemento ";I;
 80     INPUT A(I)
 90     NEXT I
100 REM Proceso de ordenacion
110   FOR I=N-1 TO 1 STEP -1
120     LET CH=0
130     FOR J=1 TO I
140        IF A(J)>A(J+1) THEN GOTO 190
150        LET T=A(J)
160        LET A(J)=A(J+1)
170        LET A(J+1)=T
180        LET CH=1
190        NEXT J
200    IF CH=0 THEN GOTO 220
210    NEXT I
220 REM Lista resultante
225   CLS : PRINT "LISTA ORDENADA" : PRINT
230   FOR I=1 TO N
240     PRINT A(I);" ";
250     NEXT I
260 END
```

Figura 4. Tabla de variables en el algoritmo de burbuja en ordenación de mayor a menor.

Lista inicial: 7, 5, 94, 0

PASO	I	J	A (J)	A (J+1)	T	CH	Lista resultante
1	3	1	7	5	–	0	7, 5, 94, 0
2	3	2	5	94	5	1	7, 94, 5, 0
3	3	3	5	0	–	1	7, 94, 5, 0
4	2	1	7	94	7	1	94, 7, 5, 0
5	2	2	7	5	–	1	94, 7, 5, 0
6	1	1	94	7	–	0	94, 7, 5, 0

Este algoritmo utiliza una variable especial, que hemos denominado *CH*, para indicar si se ha realizado o no algún intercambio. Si se ha efectuado alguno el valor de la variable es 1, en caso contrario es 0. Así, antes de dar cada vuelta se fija el valor de CH a 0, y si al final de la misma este valor no se ha alterado, podemos asegurar que la lista ya está ordenada, con lo que podemos finalizar. La ventaja de utilizar esta variable es que no damos más vueltas de las necesarias; en el momento en que la lista esté ordenada, el proceso se detiene. La utilización de la variable *CH* corresponde a una forma mejorada (más rápida) del algoritmo de burbuja. Este podría funcionar igual suprimiendo las líneas 120, 180 y 200. Compruebe que esto es cierto.

Veamos la tabla de valores de las variables para la ordenación de la lista (Fig. 4):

Tanto en este algoritmo como en el anterior, hemos realizado la ordenación de mayor a menor. Si deseamos la ordenación inversa, es decir, de menor a mayor, bastará –en ambos casos– modificar una sola instrucción del programa. Piense usted cuál será esta modificación antes de seguir leyendo. Tal como probablemente habrá imaginado, bastará cambiar la línea en la que se comparan los valores (en este caso la 140) para obtener la ordenación inversa. Así, si escribimos:

```
140   IF A(J)<A(J+1) THEN GOTO 110
```

sólo se modificará la lista cuando un elemento sea mayor que su vecino de la derecha. Si es menor no debemos efectuar ninguna variación. Por tanto, nos saltamos el intercambio, y seguimos comparando los elementos siguientes.

Vamos a aplicarlo a la ordenación de la misma lista que en el caso anterior, pero en este caso de menor a mayor. Antes de mirar el resultado, trate de escribir la tabla por su cuenta, esto le ayudará a comprender mejor el funcionamiento de este algoritmo. Nosotros le damos la tabla en la figura 5.

Figura 5. Tabla de variables en el algoritmo de burbuja en ordenación de menor a mayor.

Lista inicial: 7, 5, 94, 0

PASO	I	J	A (J)	A (J+1)	T	CH	Resultado
1	3	1	7	5	7	1	5, 7, 94, 0
2	3	2	7	94	–	1	5, 7, 94, 0
3	3	3	94	0	94	1	5, 7, 0, 94
4	2	1	5	7	–	0	5, 7, 0, 94
5	2	2	7	0	7	1	5, 0, 7, 94
6	1	1	5	0	5	1	0, 5, 7, 94

Observe el valor de la variable *CH*. Note que el valor de esta variable es 0 al empezar cada vuelta del bucle más interior, y que una vez se ha realizado un intercambio dentro de la misma vuelta su valor sólo puede ser 1. No indica, por tanto, si hay o no intercambio en cada paso (por ejemplo, en el Paso 2 no hay intercambio y sin embargo *CH* vale 1, dado que en el Paso 1 sí se han intercambiado dos elementos). Tal como hemos dicho, lo que indica el si se ha producido intercambio a lo largo de toda la vuelta, o lo que es lo mismo, para cada valor de I.

16.2.3 Algoritmo de Shell

Todos los algoritmos de ordenación se basan en comparar elementos y, si es necesario, en intercambiarlos. Evidentemente, si para obtener la lista ordenada se puede minimizar el número de comparaciones y de intercambios, el proceso será mucho más rápido. Por esta razón se buscan algoritmos mejorados, de forma que, aunque su planteo o su funcionamiento no sea el más simple e intuitivo, nos permitan obtener el resultado con el menor número de pasos posible.

Comparación e intercambio entre los elementos alejados entre sí

En este sentido, vamos a ver el *algoritmo de ordenación de Shell*, conocido también como *algoritmo de inserción con elementos decrecientes.* La idea de este algoritmo es reducir rápidamente el desorden en la lista, empezando por comparar –e intercambiar si es necesario– los elementos más alejados entre sí. La ordenación se realiza en pasos sucesivos, de forma que a cada paso se reduce el intervalo (la distancia) entre los elementos que se comparan, acabando por comparar elementos adyacentes.

Tal como hemos dicho, no resulta evidente ver por qué este método permite obtener mejores resultados que los anteriores en cuanto al tiempo de ordenación. En este sentido, no entraremos en consideraciones teóricas al respecto. Bastará únicamente decir que se comprueba que este método permite en promedio obtener la ordenación en menor tiempo.

Insistimos en que el algoritmo ofrece mejores resultados en promedio; la mayor o menor eficacia de un algoritmo en un caso concreto, dependerá de la situación de desorden en la lista que se quiera ordenar.

Podría darse la situación de que para una lista determinada, un algoritmo aparentemente más rápido no diera mejores resultados. Esto puede suceder especialmente en listas cortas. Cuando el número de elementos de la lista crece, no hay discusión; los algoritmos mejorados ofrecen ventajas evidentes.

Veamos cómo actúa el método sobre una lista dada. Supongamos que queremos ordenar la lista de números de menor a mayor:

44, 55, 12, 42, 94, 18, 06, 67

El primer elemento tomado será el 4. La ordenación de 4 en 4 produce:

44 55 12 42 94 18 06 67

44 18 06 42 94 55 12 67

Disminuimos ahora el incremento a 2. En este caso se realizan las comparaciones sucesivas de los números tal como indican las líneas de unión.

44 18 06 42 94 55 12 67

El proceso consistirá en ir ordenando los pares de números como si se tratase de dos listas distintas (Los números en posición impar por una parte: 1.°, 3.°, 5.° y 7.° y los situados en posición par por otra: 2.°, 4.°, 6.° y 8.°). De esta forma se comparan todos los elementos de cada lista hasta dejarlos ordenados. La lista de los impares quedará así

06 □ 12 □ 44 □ 94 □

Y la lista de los pares quedará así:

□ 18 □ 42 □ 55 □ 67

El resultado de la ordenación será pues:

06 18 12 42 44 55 94 67

Este proceso es el mismo que en el paso anterior, cuando hemos utilizado el incremento 4. Sin embargo, hemos detallado más este segundo paso (con el incremento a 2) para hacerlo más fácil de ver.

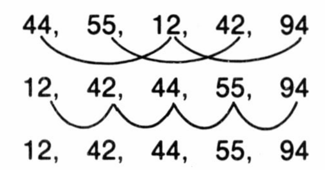
Por último, el incremento sera 1. La ordenación de 1 en 1 dará:

06 18 12 42 44 55 94 67

06 12 18 42 44 55 67 94

Tal como podemos observar, cada paso se beneficia de los anteriores, dado que disminuye el número de intercambios a efectuar.

Si el número de elementos de la lista fuera impar, por ejemplo 5, la evolución sería la siguiente:



Veamos ahora el programa para ejecutar este algoritmo:

Algoritmo de Shell

Programa del «Algoritmo de Shell»

```
NEW
 10 REM ==========================================
 20 REM ORDENACION; Algoritmo de Shell
 30 REM ------------------------------------------
 40 CLS
 50 REM Entrada de los elementos a ordenar
 60     INPUT "Numero de elementos: ";N
 70     DIM V(N)
 80     FOR I=1 TO N
 85          PRINT "Elemento ";I;
 90          INPUT V(I)
100          NEXT I
110 REM Proceso de ordenacion
120    LET GA=INT(N/2)
130    IF GA<1 THEN GOTO 270
140          FOR I=GA TO N
150              LET  J=I-GA
160              IF J<1 THEN GOTO 230
170                  IF V(J)<=V(J+GA) THEN GOTO 210
180                       LET T=V(J)
190                       LET V(J)=V(J+GA)
200                       LET V(J+GA)=T
210                  LET J=J-GA
220                  GOTO 160
230              NEXT I
240          LET GA=INT(GA/2)
250          GOTO 130
270 REM Escritura de resultados
280    FOR I=1 TO N
290          PRINT V(I);" ";
300          NEXT I
310 END
```

En primer lugar (líneas 60 a 100) introducimos el número de elementos de la lista, así como los valores de estos elementos. Se inicia entonces el proceso de ordenación (líneas 120 a 250). Definimos en primer lugar una variable (que llamamos GA) que contiene la distancia entre los números que comparamos en cada vuelta; al empezar se define

esta variable como la mitad del número de elementos que contiene la lista. Dado que este número puede ser impar, al dividirlo por dos nos daría un número con decimales. Por esta razón tomamos la parte entera de la división. Se comparan entonces y se intercambian, si es necesario, los elementos situados a esta distancia. Se vuelve entonces a partir el intervalo por la mitad, repitiéndose todo el proceso, y así hasta alcanzar el valor 1. Para ver el funcionamiento práctico del programa, realizaremos la tabla de valores correspondiente a la ordenación por este método de la lista (Fig. 6).

7, 5, 94, 0

	GA	I	J	J+GA	V (I)	V (J)	Lista resultante
Paso 1	2	2	0	–	–	–	–
Paso 2	2	3	1	3	7	94	7, 5, 94, 0
Paso 3	2	3	–1	–	–	–	–
Paso 4	2	4	2	4	5	0	7, 0, 5, 94
Paso 5	2	4	0	–	–	–	–
Paso 6	1	1	0	–	–	–	–
Paso 7	1	2	1	2	7	0	0, 7, 5, 94
Paso 8	1	2	0	–	–	–	–
Paso 9	1	3	2	3	7	5	0, 5, 7, 94
Paso 10	1	3	1	2	0	5	0, 5, 7, 94
Paso 11	1	3	0	–	–	–	–
Paso 12	1	4	3	4	7	94	0, 5, 7, 94
Paso 13	1	4	2	3	5	7	0, 5, 7, 94
Paso 14	1	4	1	2	0	5	0, 5, 7, 94
Paso 15	1	4	0	–	–	–	–

Figura 6. Tabla de variables en el algoritmo de Shell.

En la tabla de la figura 6 se detalla paso por paso el proceso que tiene lugar para ordenar la lista. Aunque aparentemente resulta más complejo, observe que de hecho sólo se efectúan intercambios en los pasos 2, 4, 7 y 9.



16.3 PROCESOS DE BÚSQUEDA

El problema de buscar un elemento dentro de una lista, se plantea muy frecuentemente en el ámbito de la programación. Vamos a ver dos

formas de solucionarlo. La primera de ellas recibe el nombre de *búsqueda secuencial*; la segunda se denomina *búsqueda binaria.*

16.3.1 Algoritmo de búsqueda secuencial

Este algoritmo sirve para buscar un valor determinado dentro de una lista. La forma de hacerlo consiste en empezar la búsqueda por el primer elemento de la lista y recorrerla –elemento a elemento del principio al fin–, comparando cada valor con el que se está buscando. El proceso finalizará o bien porque se ha encontrado el elemento que se buscaba, o bien porque se ha llegado al final de la lista sin que exista el elemento buscado.

En caso de que se haya encontrado el elemento, el programa imprimirá un mensaje al efecto, indicando la posición en que se encuentra el valor que buscamos. Veamos el listado del programa para realizar una búsqueda secuencial.

Algoritmo de búsqueda secuencial

Programa de «búsqueda secuencial»

```
NEW
 10 REM ========================================
 20 REM BUSQUEDA; Algoritmo de busqueda lineal
 30 REM ----------------------------------------
 40 CLS
 50 REM Entrada de datos
 60   INPUT "Numero de elementos: ";N
 70   DIM A(N)
 80   FOR I=1 TO N
 85      PRINT "Elemento ";I;
 90      INPUT A(I)
100      NEXT I
110   INPUT "Elemento a buscar: ";E
120 REM Proceso de busqueda
130   LET I=1
140   IF I>N THEN GOTO 200
150      IF E=A(I) THEN GOTO 180
160         LET I=I+1
170         GOTO 140
175 REM Impresion del resultado
180   PRINT "El elemento ";E;" ocupa la posicion ";I
190   GOTO 210
200   PRINT "El elemento ";E;" no esta en la lista"
210 END
```

Veamos el funcionamiento del programa. En primer lugar introducimos los datos (líneas 60 a 100), empezando por el número de elementos de la lista. Dimensionamos la variable que la contendrá *(A)*, pasando a continuación a escribir cada uno de sus elementos. Por último, habrá que introducir el dato a buscar.

Se efectúa entonces el proceso de búsqueda (líneas 130 a 170). En primer lugar asignamos a la variable *I* el valor 1. Esta variable nos indicará

la posición del elemento de la lista que estamos comparando. Así, cuando empezamos, la variable *I* vale 1, ya que vamos a comparar con el primer elemento de la lista. A continuación empezará el proceso de búsqueda. En primer lugar controlamos si se ha llegado al final de la lista, es decir, si *I* es mayor que el número de elemento que tenemos *(N).*

Seguidamente, se compara el elemento correspondiente de la lista (en posición *I*) con el valor que buscamos. Si no existe igualdad, debemos ir a comparar con el siguiente elemento; para ello incrementamos el valor de *I*, y volvemos a la línea 140. En el momento que encontremos un elemento igual al que buscamos, finalizará el proceso. Por otra parte, si el valor buscado no se encuentra en la lista, llegaremos al final de la misma, compararemos con el último valor (que será distinto), e incrementaremos *I*, con lo que ésta será superior a *N* y habremos terminado.

Veamos la tabla de variables para un proceso concreto (Fig. 7).

Supongamos que deseamos buscar en la lista:

7, 5, 94, 0

un valor cualquiera, por ejemplo el 4. La tabla será:

	N	**I**	**E**	**A (I)**	**Resultado**
Paso 1	4	1	4	7	distintos
Paso 2	4	2	4	5	"
Paso 3	4	3	4	94	"
Paso 4	4	4	4	0	"

Figura 7. Tabla de variables en el algoritmo de búsqueda secuencial de un elemento que no está en la lista.

El proceso finaliza porque I alcanza un valor mayor que N; no hemos encontrado por tanto ningún elemento igual. En pantalla aparecerá el mensaje:

El elemento 4 no está en la lista

Vamos a ver cómo evolucionará la tabla de variables para un elemento que sí está en la lista; por ejemplo, el 94. (Fig. 8).

	N	**I**	**E**	**A (I)**	**Resultado**
Paso 1	4	1	94	7	distintos
Paso 2	4	2	94	5	"
Paso 3	4	3	94	94	iguales

Figura 8. Tabla de variables en el algoritmo de búsqueda secuencial de un elemento que está en la lista.

El valor buscado existe en la lista; aparecerá por tanto en pantalla:

El elemento 94 ocupa la posición 3

En este caso la lista que hemos introducido es muy corta, y no importa tener que escribirla cada vez. Sin embargo, podemos modificar el programa de forma que podamos realizar más de una búsqueda sin que haga falta introducir la lista cada vez. Así tendremos:

```
210  INPUT "Desea continuar (S/N): ";R$
220  IF R$="S" OR R$="s"  THEN GOTO 110
230 END
```

Este algoritmo sirve para realizar una búsqueda en una lista cualquiera. No es necesario que los elementos sean del mismo tipo. Podemos tener una lista con letras y números mezclados; en este caso trabajaremos con una variable de cadena, que deberá contener la lista de valores (A$ en lugar de A), también deberemos usar una variable de cadena para el elemento a buscar (*E$* en lugar de *E*).

Por otra parte, este algoritmo no presupone nada respecto a la situación de los elementos que la componen, no hace falta que estos estén ordenados.

El algoritmo de búsqueda secuencial es lento

Sin embargo este algoritmo es bastante lento, ya que debemos recorrer toda la lista antes de poder asegurar que un elemento no está en ella. Si disponemos de una lista ordenada, se pueden desarrollar procesos de búsqueda más eficientes. En la siguiente sección veremos uno de estos sistemas, de uso muy frecuente, que se denomina algoritmo de búsqueda binaria.

16.3.2 Algoritmo de búsqueda binaria

Este algoritmo sirve para realizar búsquedas en una lista ordenada.

La idea básica que sigue consiste en partir la lista por la mitad, y mirar en qué lado se encuentra el elemento que buscamos. Comparamos para ello el valor buscado con el elemento central obtenido. Si este es mayor que el que buscamos, nos quedamos con la parte izquierda de la lista, si es menor, nos quedamos con la parte derecha. Con el trozo de lista que nos hemos quedado, repetimos la operación: partimos, comparamos, etc. Y esta operación se va repitiendo hasta que se encuentra un elemento igual al que buscamos, o bien hasta que no podemos seguir dividiendo la lista, pues nos hemos quedado con un sólo elemento.

Este proceso lo efectuamos en la vida corriente en muchas ocasiones sin ser conscientes de ello. Por ejemplo, para buscar un nombre en un listín telefónico, o una palabra en un diccionario, etc.

Normalmente abrimos el diccionario por una página cualquiera, y miramos qué letra sale. Comparamos ésta con la palabra que buscamos, y si la letra encontrada es menor (es decir, está antes en el abecedario), la siguiente búsqueda la hacemos sólo en el trozo derecho del diccionario. De nuevo partimos éste y vamos repitiendo la operación hasta encontrar la página en la que se encuentra la palabra que

buscamos. En este punto generalmente suele hacerse una búsqueda lineal; es decir, empezamos por el principio de la página hasta encontrar la palabra que queríamos; o bien constatamos que la palabra buscada no se encuentra en el diccionario.

Evidentemente, no podríamos hacerlo así si el diccionario no estuviera ordenado alfabéticamente, tal como hemos dicho. Esta condición es indispensable. Antes de utilizar este procedimiento de búsqueda binaria debemos asegurarnos de que la lista en la que buscamos está correctamente ordenada, pues de lo contrario llegaríamos a conclusiones erróneas.

Veamos el listado del programa para realizar una búsqueda binaria en una lista dada de números, que introduciremos, ordenada de menor a mayor.

Algoritmo de búsqueda binaria

Programa de «algoritmo de búsqueda binaria»

```
NEW
 10 REM =========================================
 20 REM BUSQUEDA; Algoritmo de busqueda binaria
 30 REM -----------------------------------------
 40 CLS
 50 REM Introduccion de los datos
 60   INPUT "Numero de elementos de la lista: ";N
 70   DIM A(N)
 80   FOR I=1 TO N
 85     PRINT "Elemento: ";I;
 90     INPUT A(I)
100     NEXT I
110   INPUT "Valor a buscar: ";X
120 REM Proceso de busqueda
130   LET I=1
140   LET D=N
150   IF I>D THEN GOTO 240
160     LET M = INT((I+D)/2)
170     IF A(M)=X THEN GOTO 210
180     IF A(M)>X THEN LET D=M-1
190     IF A(M)<X THEN LET I=M+1
200     GOTO 150
210 REM Impresion del resultado
220   PRINT "El elemento ";X;" ocupa la posicion ";M
230   GOTO 250
240   PRINT "El elemento ";X;" no esta en la lista"
250   INPUT "Desea continuar (S/N): ";R$
260   IF R$="S" OR R$="s" THEN GOTO 110
270 END
```

En primer lugar, realizamos como siempre la introducción de las variables: número de elementos de la lista, valores que la componen, y elemento a buscar *(X)*. A continuación se inicia el proceso de búsqueda. Empezamos definiendo dos variables, que llamamos *I* (izquierda) y *D* (derecha). Estas variables serán las que nos indicarán los límites de la lista en la que efectuamos la búsqueda; así, para empezar, la variable *I* vale 1, y la variable *D* vale *N*, pues en el primer paso consideramos la

lista completa. A continuación se define la condición de repetición del proceso, que es que la variable *I* sea menor que la variable *D*, lo cual quiere decir que tenemos al menos un elemento en la lista.

Si, por el contrario, esto no es así significa que no tenemos lista donde hacer la búsqueda, con lo que ésta finaliza. Esto puede pasar porque hemos entrado una lista de 0 elementos, o bien porque –tal como veremos– al ir desarrollando el proceso de búsqueda hemos ido desplazando los límites hasta quedarnos sin ningún elemento para comparar. En este caso podemos afirmar que el elemento que buscamos no está en la lista y finalizar.

Si se cumple la condición de que el límite izquierdo de la lista *(I)* es inferior al límite derecho *(D)* proseguimos el proceso. En primer lugar, partimos la lista por la mitad, o lo que es lo mismo, determinamos la posición del elemento central de la lista, y asignamos el resultado a la variable *M* (medio).

Veamos entonces qué posibilidades tenemos:

Puede pasar que el elemento que hemos obtenido (el que ocupa la posición *(M)* sea el que buscábamos; en este caso finalizaremos el proceso, dando el correspondiente mensaje.

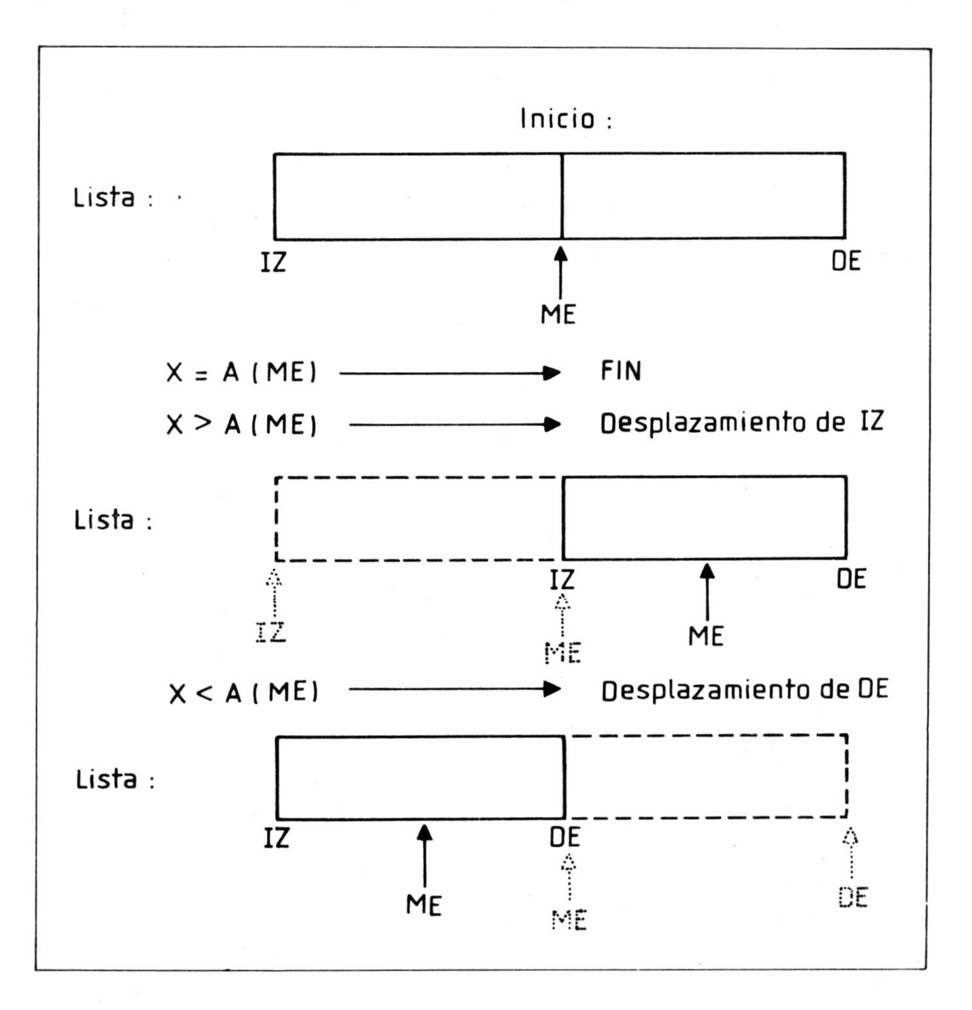


Figura 9. Procedimiento de búsqueda binaria.

Por otra parte, puede ser que este elemento (el que ocupa la posición central) sea superior al que buscamos; por tanto, dado que la lista está ordenada, podemos asegurar que el valor X estará en el primer trozo de la lista, con lo que podemos desplazar el límite derecho de la misma al punto medio, prescindiendo de todos los elementos situados a la derecha de éste.

Si, por el contrario, el valor obtenido es inferior al que buscamos, prescindiremos de la parte izquierda de la lista, desplazando el límite izquierdo al punto medio de la misma.

Esta operación se repetirá, tal como hemos dicho, tantas veces como sea necesario, o bien hasta encontrar el valor buscado, o bien hasta que nos quedemos sin lista que partir. En este caso, daremos también el oportuno mensaje.

En la figura 9 se representa en forma gráfica el proceso que tiene lugar dentro del bucle, entre las líneas 150 y 200.

Vamos a ver la tabla de variables (Fig. 10) para buscar el número 12 en la lista de números:

0, 5, 7, 94

Figura 10. Tabla de variables en el algoritmo de búsqueda binaria de un elemento que no está en la lista.

	X	**I**	**D**	**M**	**A (M)**	**Lista de búsqueda**
Paso 1	12	1	4	2	5	0, 5, 7, 94
Paso 2	12	3	4	3	7	0, 5, 7, 94
Paso 3	12	4	4	4	94	0, 5, 7, 94
Paso 4	12	4	3	–	–	–

Tal como se puede ver en la tabla, vamos partiendo la lista:

En el primer paso tomamos todos sus elementos.

En el segundo sólo consideramos a partir del tercero hasta el final.

En el paso 3 tomamos únicamente el elemento cuarto, y dado que este valor no coincide con el que buscamos, al mover el límite derecho de la lista, éste pasa a ser menor que el izquierdo, con lo cual finaliza el proceso; podemos asegurar que el elemento buscado no se encuentra en la lista.

Veamos ahora la búsqueda de un elemento que forme parte de la lista, por ejemplo, el número 24 en la lista:

–3, 0, 1, 5, 12, 24, 90

La tabla de valores será ahora la que se muestra en la figura 11.

Figura 11. Tabla de variables en el algoritmo de búsqueda binaria de un elemento que está en la lista.

	X	**I**	**D**	**M**	**A (M)**	**Lista de búsqueda**
Paso 1	24	1	7	4	5	–3, 0, 1, 5, 12, 24, 90
Paso 2	24	3	7	5	12	–3, 0, 1, 5, 12, 24, 90
Paso 3	24	6	7	6	24	–3, 0, 1, 5, 12, 24, 90

Este método con respecto a la búsqueda secuencial presenta la ventaja de reducir el número de comparaciones; supone una gran mejora cuando se trata de realizar búsquedas en listas de gran número de elementos, ya que el número de pasos necesarios para obtener el resultado es mucho menor. Sin embargo, repetimos una vez más que este algoritmo *precisa que la lista de búsqueda esté ordenada.*

RESUMEN

Un algoritmo puede definirse como una secuencia ordenada de pasos y de decisiones concretas, exenta de ambigüedades, que lleva a la solución de un problema.

Para construir un algoritmo se debe buscar el equilibrio entre conseguir un procedimiento que sea general (útil en muchos casos), y que al mismo tiempo sea eficaz (lo más simple y rápido posible).

Para resolver problemas que se presentan frecuentemente en programación, se han desarrollado distintos algoritmos.

Uno de estos problemas es la ordenación de listas, ya sean numéricas o textuales. Existen distintos procedimientos para ello. Todos se basan en comparar los elementos entre sí, y si es necesario, intercambiarlos, la diferencia consiste en el orden que se sigue.

El algoritmo de *selección* va comparando cada elemento de la lista, avanzando uno a uno de izquierda a derecha, con los que se encuentran más a la derecha de él, y realizando el intercambio, si es necesario.

El algoritmo de *burbuja* actúa de forma que los elementos más pequeños (más ligeros) van avanzando hacia el final de la lista (la superficie), quedando los mayores (más pesados), al principio de esta (el fondo). Para ello cada elemento se compara con su vecino, empezando por la izquierda, y se intercambia si es necesario, de forma que el más pequeño queda al final de la lista. La segunda vez se repite la operación pero sólo hasta el penúltimo elemento. Este proceso se repite hasta llegar al primer elemento de la lista.

Existe una forma mejorada de este algoritmo que consiste en comprobar si en una de las vueltas no se ha realizado ningún intercambio. En este caso se puede afirmar que la lista ya está ordenada.

Por último, el algoritmo de *Shell*, o algoritmo de *inserción* con elementos decrecientes, se basa en comparar los elementos más alejados entre sí de la lista, de cara a reducir rápidamente el desorden de la misma. Para ello se va dividiendo la lista en intervalos sucesivamente más pequeños, hasta llegar a comparar elementos adyacentes.

EJERCICIOS DE AUTOCOMPROBACIÓN

Completar las siguientes frases:

1. Un algoritmo puede definirse como una .... ordenada de pasos, exenta de ambigüedades.

2. Un algoritmo trata de resolver un problema de forma ....

3. El número de pasos de un algoritmo debe ser ....

4. Los algoritmos de clasificación sirven para .... listas.

5. El algoritmo de selección es un algoritmo de ....

6. El algoritmo basado en comparar cada elemento de la lista de izquierda a derecha con todos los que están más a su derecha en la lista se denomina algoritmo de ....

7. El algoritmo de burbuja es un algoritmo que sirve para .... listas.

8. El algoritmo de burbuja va comparando elementos .... de la lista.

9. El algoritmo de Shell sirve para .... listas.

10. El algoritmo que empieza comparando los elementos más alejados de la lista, para ir reduciendo el intervalo sucesivamente hasta llegar a comparar elementos adyacentes, se denomina algoritmo de ....

Encierre en un círculo la letra que corresponde a la alternativa correcta:

11. Un algoritmo es:

a) Un sistema de decisión entre alternativas.

b) Un sistema capaz de solucionar cualquier problema.

c) Una secuencia finita de pasos, exenta de ambigüedades, orientada a resolver un problema concreto.

d) Un sistema que sirve únicamente para ordenar listas, ya sean numéricas o textuales.

12. Para resolver correctamente un problema hay que:

a) Buscar la solución para cada caso concreto.

b) Buscar un algoritmo general.

c) Introducirlo en el ordenador.

d) Traducirlo a código máquina.

13. Indique qué característica *NO* corresponde a un buen algoritmo

a) Ambiguo.

b) General.

c) Simple.

d) Rápido.

14. Un algoritmo es efectivo cuando:

a) No tiene solución.

b) El número de pasos en finito.

c) Ordena tablas o listas.

d) No acaba nunca.

15. Un algoritmo está exento de ambigüedades cuando:

a) Se realiza en sucesivas etapas.

b) Empieza por el principio.

c) No da errores, aunque existan muchas posibilidades de que los haya.

d) Considera todas las alternativas, resolviendo de forma concreta cada una.

16. Un algoritmo de ordenación sirve para:

    a) Obtener listas infinitas.

    b) Mezclar listas numéricas y textuales.

    c) Ordenar listas de longitud finita, ya sean numéricas o textuales.

    d) Introducir el alfabeto en el ordenador.

17. El algoritmo de selección es:

    a) Un algoritmo de ordenación de listas.

    b) Un sistema de almacenar datos.

    c) Un sistema de seleccionar los elementos erróneos de una lista.

    d) Un algoritmo para invertir listas.

18. Un algoritmo de ordenación:

    a) Intercambia todos los elementos sin compararlos.

    b) Compara los elementos de dos listas.

    c) Anula los elementos menores.

    d) Compara los elementos de la lista entre sí, intercambiándolos si es necesario.

19. El algoritmo de burbuja:

    a) Compara los elementos adyacentes de la lista, haciendo avanzar los menores hacia el final.

    b) Ordena listas siempre que sus elementos sean pequeños.

    c) Intercambia valores de dos listas.

    d) No utiliza variables de control.

20. El algoritmo de Shell:

    a) Es un sistema de reducir errores.

    b) Es un algoritmo de intercambio.

    c) Compara primero los elementos más alejados de la lista para reducir rápidamente el desorden.

    d) Es un algoritmo muy complejo, para resolver problemas numéricos.

16.4 INTERCALACIÓN O FUSIÓN

Vamos a ver ahora un procedimiento para resolver otro problema que se nos puede presentar: obtener una *lista única y ordenada, a partir de otras también ordenadas*. Este procedimiento se conoce generalmente con el nombre de *Simple Merge*, lo que podríamos traducir por *mezcla simple*, y también con el nombre de *método de intercalación* o *método de fusión*.

Supongamos que tenemos dos listas ordenadas, a partir de las cuales deseamos obtener una lista única. Un ejemplo de este problema sería el que se le plantearía a un profesor con dos grupos de alumnos, que en un momento dado pasen a formar un sólo grupo. El profesor tiene las listas de las dos clases por separado, ordenadas alfabéticamente, y le interesara a partir de este momento tener una lista única con todos los alumnos de los dos grupos juntos, y también en orden alfabético. Veamos cuál será el proceso a seguir para conseguir su objetivo.

Para obtener la lista completa, tomará las dos listas de las clases por separado, y empezará comparando el primer nombre de la primera lista con el primer nombre de la segunda. Elegirá el que alfabéticamente tenga un valor más bajo, lo tachará en la lista que corresponda, y lo escribirá en la nueva lista.

Tomará ahora las dos listas y comparará de nuevo los dos primeros elementos (evidentemente sin considerar el que ya habíamos tachado). De nuevo sacará el que tenga el valor más bajo y lo pasará a la nueva lista. Este proceso se irá repitiendo hasta que se hayan tachado todos los elementos de una lista. En este punto bastará que añada los elementos sin tachar que queden en la otra lista, al final de la nueva. Con esto habrá obtenido una lista única y ordenada con los nombres de los alumnos de las dos clases. En la figura 12 se representa este proceso.

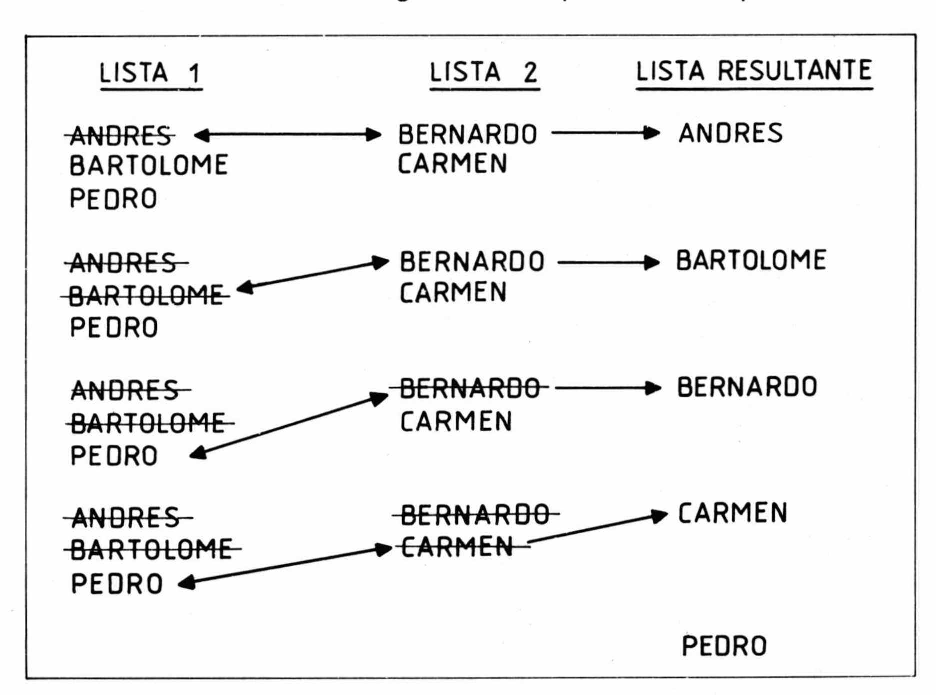


Figura 12. Procedimiento de Simple Merge.

Veamos ahora el listado del programa para intercalar dos listas de números.

Algoritmo Simple Merge

Programa del «algoritmo Simple Merge»

```
NEW
10 REM ======================================
20 REM INTERCALACION; Algoritmo Simple Merge
30 REM --------------------------------------
40    CLS
50 REM introduccion de los datos
60    INPUT "Numero de elementos de la 1a. lista: ";N
70    PRINT "INTRODUZCA LA PRIMERA LISTA"
80    DIM A$(N)
90    FOR I=1 TO N
100      PRINT "Elemento ";I;  :  INPUT A$(I)
110      NEXT I
115   CLS
120   INPUT "Numero de elementos de la 2a. lista: ";M
130   PRINT "INTRODUZCA LA SEGUNDA LISTA"
140   DIM B$(M)
150   FOR I=1 TO M
160      PRINT "Elemento ";I;  : INPUT B$(I)
170      NEXT I
180   LET LR=N+M : DIM C$(LR)
190 REM Proceso de intercalacion
200   LET I=1 : LET J=1 : LET K=1
210   IF I>N OR J>M THEN GOTO 300
220      IF A$(I) > B$(J) THEN GOTO 260
230         LET C$(K)=A$(I)
240         LET I=I+1
250         GOTO 280
260         LET C$(K)=B$(J)
270         LET J=J+1
280      LET K=K+1
290      GOTO 210
300   IF J>M THEN GOTO 370
320      FOR R=J TO M
330            LET C$(K)=B$(R)
340            LET K=K+1
350            NEXT R
360            GOTO 410
370      FOR R=I TO N
380            LET C$(K)=A$(R)
390            LET K=K+1
400            NEXT R
410  REM Impresion del resultado
420    CLS : PRINT "LISTA RESULTANTE"
430       FOR I=1 TO LR
440             PRINT C$(I)
450             NEXT I
460       END
```

En primer lugar se efectúa la introducción de las dos listas (líneas 60 a 180). Se empieza pidiendo el número de elementos de la primera lista *(N)*, se dimensiona la variable que la deberá contener *(A$)*, y se introducen los elementos correspondientes.

Recuerde que la instrucción de dimensionamiento puede variar según el ordenador en que se efectúe, por lo que deberá tener en cuenta el capítulo de Prácticas con el Ordenador correspondiente para ver cómo funciona en su máquina.

A continuación se pide el número de elementos de la segunda lista *(M)*, se dimensiona la variable correspondiente *(B$)*, y se introducen sus valores.

Una vez sabemos el número de elementos que hay en una y otra lista, podemos saber cuántos habrán en la lista resultante, que se almacenará en la variable *C$*. La dimensión de esta variable será, pues, la suma de las dimensiones de las otras dos *(LR)*.

Se inicia entonces el proceso de intercalación o fusión. Tal como hemos dicho, la «mezcla» se hace mientras las dos listas tengan algún elemento. Utilizaremos tres variables para indicar la posición del elemento que consideramos en cada una de las tres listas. Al empezar estos indicadores o punteros señalarán el primer elemento; por tanto les asignamos el valor 1 (línea 200).

Cada vez que «tachamos» un elemento de la lista y lo trasladamos a la nueva, incrementamos el valor del puntero correspondiente; es decir, avanzamos en aquella lista. El proceso finaliza cuando hemos tomado el último elemento de alguna de las dos listas (línea 210).

Mientras tanto, se realiza el proceso de fusión (líneas 220 a 290). Para ello comparamos los elementos que correspondan (los que indican las variables I y J) de cada lista. El menor de ellos se escribe en la lista de salida, y se avanza al siguiente elemento de la lista que lo contenía, incrementando el puntero correspondiente (línea 240 para la primera lista, y líneas 340 para la segunda). Simultáneamente, habrá que incrementar el puntero de la lista de salida, dado que hemos añadido un elemento, tanto si éste era de la primera lista como de la segunda (línea 280).

Cuando se acabe una de las dos listas, añadiremos lo que queda de la otra al final de la lista de salida (líneas 300 a 400). Para saber en cuál de las dos listas quedan elementos se compara el valor actual del puntero correspondiente con el número de elementos de esta lista; así, si el puntero de la segunda lista es mayor que el número de elementos máximo de ésta (línea 300), indica que la segunda lista ha llegado al final, con lo que habrá que añadir a la lista de salida los elementos que queden de la primera lista (líneas 370 a 400); por el contrario, si el puntero no es mayor que el máximo, significa que es en esta lista en la que quedan elementos. Por tanto, los añadiremos al final de la lista de salida (líneas 320 a 350).

Finalmente, no queda más que imprimir el resultado, es decir la lista completa ordenada (líneas 420 a 450).

Veamos la tabla de variables correspondientes a realizar la fusión de las dos listas siguientes:

1.ª LISTA	2.ª LISTA	
FERNANDEZ	BURGOS	TRIGUERO
GARCIA	FRANCISCO	VIDAL
HERRERO	NAVARRO	ZALDIVAR
MARTIN	SANTOS	

El proceso se indica en la siguiente tabla (Figura 13), la primera parte de la cual corresponde al proceso de mezcla o fusión, la segunda añade a la lista resultante los elementos que quedan de la más larga de las dos listas de entrada.

I	J	K	A$ (I)	B$ (J)	C$ (K)
1	1	1	FERNANDEZ	BURGOS	BURGOS
1	2	2	FERNANDEZ	FRANCISCO	FERNANDEZ
2	2	3	GARCIA	FRANCISCO	FRANCISCO
2	3	4	GARCIA	NAVARRO	GARCIA
3	3	5	HERRERO	NAVARRO	HERRERO
4	3	6	MARTIN	NAVARRO	MARTIN
5	3	7	–	–	–
R		K	–	B$ (R)	C$K)
3		7	–	NAVARRO	NAVARRO
4		8	–	SANTOS	SANTOS
5		9	–	TRIGUERO	TRIGUERO
6		10	–	VIDAL	VIDAL
7		11	–	ZALDIVAR	ZALDIVAR

Figura 13. Tabla de variables en la fusión de dos listas.

En el momento que incrementamos la variable I, esta toma un valor superior al número de elementos de la primera lista. La primera parte del proceso se da por terminada, quedará entonces añadir el resto de la segunda lista al final de la lista resultante, tal como indica la segunda parte de la tabla.



16.5 OTRO EJEMPLO PRÁCTICO: EL CALENDARIO

Para finalizar este capítulo vamos a plantearnos el problema de confeccionar un calendario.

Aunque este ejemplo no tiene una utilidad general como los anteriores problemas que hemos visto, nos va a servir para ilustrar el correcto seguimiento de un algoritmo.

Para confeccionar el calendario, escribiremos los meses por separado, indicando el nombre del mes del que se trate, así como los nombres abreviados de los días de la semana, con los números colocados en filas en la posición correspondiente, tal como se representa en la figura 14.

El primer problema que se nos plantea es calcular en qué día de la semana cae el 1 de enero de un año dado. Para ello utilizaremos el algoritmo de Zeller, que sirve en general para determinar en qué día de

Figura 14. Formato del calendario.

ENERO

LU	MA	MI	JU	VI	SA	DO
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31	–	–	–

la semana cae un día cualquiera, de cualquier año. El enunciado es el siguiente:

Variables que se utilizan:

Mes del año: Se utiliza un número para indicar el mes del año del que se trate. Sin embargo, este número no es el que se emplea normalmente, sino que se utiliza la siguiente codificación:

Enero: 11
Febrero: 12
Marzo: 1
Abril: 2
Mayo: 3
Junio: 4
Julio: 5
Agosto: 6
Septiembre: 7
Octubre: 8
Noviembre: 9
Diciembre: 10

Esto implica además que el año empiece en marzo, y finalice en febrero. Así, el mes de enero de 1916 se introducirá como mes 11 del 1915. De todas formas, al escribir el programa, lo haremos de tal forma que podamos escribir el mes y el año en la forma que normalmente lo hacemos; el propio programa se encargará de las oportunas manipulaciones.

Día del mes: Indica el número del día, tal como se utiliza normalmente.

Año de la centuria: Corresponde a las dos últimas cifras del número que indica el año; así, para el año 1916, el número a utilizar será el 16.

Centuria: Corresponde a las dos primeras cifras del número que indica el año; así, para el año 1916 el número a utilizar será el 19.

El proceso a realizar es el siguiente:

- Multiplicar el mes del año por 13, restarle 1 y el resultado dividirlo por 5, tomando la parte entera del resultado.
- Dividir el año de la centuria por 4, y tomar la parte entera.
- A continuación, dividir la centuria por 4, y tomar también la parte entera.
- Sumar ahora los tres resultados obtenidos, más el día del mes, más el año de la centuria, y restar al resultado el doble de la centuria.
- Con el valor obtenido, efectuar la división por 7, y tomar el resto de esta división. Con ello determinaremos el día de la semana de la siguiente forma:

S el resto es 0 el día es DOMINGO

Si el resto es 1 el día es LUNES

Si el resto es 2 el día es MARTES

Si el resto es 3 el día es MIÉRCOLES

Si el resto es 4 el día es JUEVES

Si el resto es 5 el día es VIERNES

Si el resto es 6 el día es SÁBADO

Tal como vemos, el algoritmo realiza con las variables unas determinadas operaciones, fundamentalmente manipulaciones matemáticas muy simples, que nos llevarán a determinar el día de la semana. Para ver con más claridad el proceso matemático a seguir, utilizaremos el nombre correspondiente a cada una de las variables que intervienen. Así, tendremos:

Mes del año: *M*

Día del mes: *D*

Año de la centuria: *Y*

Centuria: *C*

No olvide utilizar el código visto para los meses.

El algoritmo expresado en forma matemática será:

$$A = INT \frac{13 \times M - 1}{5}$$

$$B = INT \frac{Y}{4}$$

$$Z = INT \frac{C}{4}$$

$$X = A + B + Z + D + Y - 2 \times C$$

R = Resto de dividir X por 7.

Con esto tenemos una completa descripción del algoritmo. Intente ahora escribir el programa para ejecutarlo. Este programa, tal como hemos dicho, será la primera parte del programa completo para la ejecución del calendario.

Veamos el listado:

Algoritmo de Zeller

Programa utilizando el algoritmo de «Zeller»

```
NEW
 10 REM ===================================
 20 REM Determinacion del dia de la semana
 25 REM utilizando el Algoritmo de Zeller.
 30 REM -----------------------------------
 35 REM Lectura de los nombres de los meses
 37 REM y de los dias de la semana
 40    DIM M$(12),D$(7)
 50    FOR I = 1 TO 12
 60       READ M$(I)
 70       NEXT I
 80    FOR I = 1 TO 7
 90       READ D$(I)
100       NEXT I
110    CLS
115 REM Entrada de la fecha
117    INPUT "Dia del mes: ";D
120    INPUT "Mes del ano";P$
140    LET I = 1
150    IF I>12 THEN PRINT "MES ERRONEO" : GOTO 120
160       IF P$=M$(I) THEN GOTO 190
170          LET I = I+1
180          GOTO 150
190    LET M = I
210    INPUT "Ano: ";A$
215 REM Preparacion de los datos
220    LET C = VAL(LEFT$(A$,2)) : LET Y = VAL(RIGHT(A$,2))
230    IF M>10 THEN LET Y = Y-1
235 REM Proceso de calculo
240    LET A = INT((13*M-1)/5)
250    LET B = INT(Y/4)
260    LET Z = INT(C/4)
270    LET X = A+B+Z+D+Y-2*C
280    LET R = X-INT(X/7)*7+1
285 REM Impresion del resultado
290    PRINT "El ";D;" de ";P$;" del ano ";A$;" es ";D$(R)
315 END
3000 DATA "MARZO","ABRIL","MAYO","JUNIO"
3010 DATA "JULIO","AGOSTO","SEPTIEMBRE","OCTUBRE","NOVIEMBRE"
3020 DATA "DICIEMBRE","ENERO","FEBRERO"
3030 DATA "DOMINGO","LUNES","MARTES"
3040 DATA "MIERCOLES","JUEVES","VIERNES","SABADO"
```

Veamos el funcionamiento de este programa. En primer lugar, hacemos la lectura de los nombres de los meses y de los nombres de los días de la semana, (líneas 50 a 100). Esto nos permitirá, tal como veremos, introducir el nombre del mes, y por otra parte, obtener como resultado el nombre del día de la semana directamente.

A continuación se solicita la fecha; tal como está ahora el programa, se podrá escribir directamente el número del día, el nombre del mes y el número del año. Se empezará escribiendo el número del día, a continuación el nombre completo del mes en mayúsculas, y a

continuación el año completo (es decir 1916, o 1999, etc.). Si el nombre del mes que se escribe no es correcto, el programa lo detecta, y solicita que se le introduzca de nuevo. Para ello, se va comparando (líneas 150 a 180) el nombre que se ha escrito (contenido en la variable *P$*) con todos los nombres de los meses, contenidos en la lista *M$*), que previamente hemos leído. A cada comparación se incrementa un contador, que hemos llamado *I*. Si este contador llega a ser mayor que 12, indica que el nombre que hemos escrito es incorrecto, ya que no se encuentra en nuestra lista ningún mes con ese nombre. Si, por el contrario, llega un momento en que se produce la igualdad, el valor del contador *I* nos indica el número correspondiente al mes entrado (línea 190), según el código establecido en el algoritmo. Es por esta razón por la que hemos escrito los meses empezando en marzo, ya que el número 1 corresponde a este mes.

Deberá ahora separarse lo que hemos llamado año de la centuria y centuria. Por esto introducimos el año en una variable de cadena, que nos servirá para separar las dos primeras cifras (será la centuria) y las dos últimas (será el año de la centuria), tal como se indica en la línea 220. Las funciones que se utilizan para ello difieren según el ordenador con el que se trabaje. Para saber cómo hacerlo en su ordenador, tenga en cuenta lo dicho en el capítulo de Prácticas con el microordenador.

A continuación se realiza el cambio correspondiente en el valor del año de la centuria, de acuerdo con el código de meses utilizado, que hace que el año comience en marzo, no en enero. Así, si estamos en este mes o en febrero, dado que según el algoritmo corresponden al año anterior, restaremos una unidad a la variable *Y* que lo contiene, tal como se hace en la línea 230.

Podremos ahora iniciar el proceso de cálculo, siguiendo estrictamente lo indicado en el algoritmo. Para determinar la parte entera de las divisiones, utilizaremos la función INT, que ya vimos en su día.

Por otra parte, para calcular el resto podemos utilizar la función MOD, si nuestra máquina la tiene, o bien realizar el cálculo tal como se hace en la línea 280. Si nuestro ordenador tiene la función MOD, podríamos escribir:

```
280 LET R = MOD(X,7) +1
```

Finalmente, escribiremos el nombre del día de la semana utilizando el valor calculado como subíndice de la lista de los nombres de los días, que habíamos leído al principio del programa en el orden conveniente.

Ahora, si utilizamos este programa podremos saber en qué día cae el 1 de enero de cualquier año, de cara a hacer el calendario completo. Para ello bastará introducir el año; los demás datos (mes y día) serán fijos.

Antes de pasar a la confección del calendario, se nos presenta, sin embargo, otro problema. Sabemos que algunos meses tienen 31 días: enero, marzo, mayo, julio, agosto, octubre y diciembre; otros tienen 30: abril, junio, septiembre y noviembre; sin embargo el mes de febrero no tiene siempre los mismos días. En los años bisiestos este mes tiene 29 días, mientras que en los no bisiestos tiene 28. Para escribir pues el

calendario de cualquier año, deberemos saber previamente si es o no bisiesto, de cara a asignar a cada mes el número correcto de días. Para conocer si un año es o no bisiesto existe también un procedimiento, según el cual, un año es bisiesto si es múltiplo de 4, excepto si lo es también de 100 (cabecera de siglo), y a su vez también múltiplo de 400.

Veamos como construir un programa que nos resuelva este problema. Este programa, como el anterior, pasará a formar parte del programa completo para escribir el calendario.

```
NEW
 10 REM ============================================
 20 REM Determinacion de si un ano es o no bisiesto
 30 REM --------------------------------------------
 40   CLS
 50   INPUT "Entre el ano: ";A
 60   LET R1 = A-INT(A/4)*4
 70   LET R2 = A-INT(A/100)*100
 80   LET R3 = A-INT(A/400)*400
 90   IF R1=0 AND R2<>0 AND R3<>0 THEN GOTO 120
100   PRINT "El ano ";A;" no es bisiesto"
110   GOTO 130
120   PRINT "El ano ";A;" es bisiesto"
130 END
```

Veamos el funcionamiento del programa. En primer lugar introducimos el año. A continuación pasamos a determinar si este número es múltiplo de 4, de 100 y de 400. Para ello, calculamos el resto de su división por 4 *(R1)*, por 100 *(R2)* y por 400 *(R3)*, en las líneas 60 a 80. Si un número es múltiplo de otro, al dividirlo por éste nos dará de resto cero.

Pasamos entonces a definir la condición: tal como hemos dicho; el año es bisiesto si es múltiplo de 4 (*R1* es cero), y simultáneamente no lo es de 100 ni de 400 (*R2* y *R3* son distintos de cero), tal como se indica en la línea 90.

Con este problema resuelto, ya podemos plantearnos la confección del calendario.

En primer lugar debemos pensar en utilizar los dos programas que acabamos de ver, dentro de este nuevo programa. Para ello los convertiremos en subrutinas, dándoles la numeración adecuada, y efectuando los cambios que sean necesarios.

Así, la rutina de cálculo del día de la semana la utilizaremos para saber en qué posición estará el primer día de cada mes. Por tanto, suprimiremos la fase de entrada de datos, ya que la variable que corresponde al día será siempre 1, y el año y el mes los tendremos fijados cuando llamemos a la rutina. La fase de cálculo será la misma, pero el resultado no lo imprimirá, sino que lo almacenaremos en una variable que utilizaremos en el programa principal, tal como veremos en el listado.

Por otra parte, una vez sepamos de qué año debe ser el calendario, podremos averiguar si es o no bisiesto; es decir, cuántos días tendrá el mes de febrero. Para ello suprimiremos la entrada de datos también en

el segundo programa, así como la fase de impresión de resultados, numerándolo convenientemente y convirtiéndolo en una subrutina.

Nos quedará ahora la confección del programa principal, el que efectivamente escribirá el calendario. En una primera fase, este programa deberá averiguar una serie de datos. En primer lugar se deberá leer el año del que se desea el calendario, que será introducido por el usuario. Por otra parte, el programa leerá los valores de las variables que sean necesarios, como el nombre de los meses, el nombre de días de cada mes (excepto de febrero, que deberá calcularlo tal como hemos dicho), etc. Para ello construiremos el correspondiente almacén de datos, así como las rutinas necesarias para su lectura.

Se procederá después a efectuar un proceso que se repetirá 12 veces: escribir el calendario de cada mes. Se empezará escribiendo el nombre del mes correspondiente, así como los días de la semana. A continuación se empezarán a escribir por filas los números, teniendo en cuenta que la posición del día 1 será variable, y vendrá dada por la rutina correspondiente. Se irán escribiendo entonces filas de números de 7 en 7, hasta llegar al mes completo. Este proceso repetitivo se incluirá dentro de un bucle, una vez finalizado éste, el programa habrá terminado. Veamos ahora el listado correspondiente a este programa.

```
  10 REM ==============================================
  20 REM Confeccion del calendario de un ano cualquiera
  30 REM ----------------------------------------------
  40    CLS
  50    DIM M$(12) : DIM D$(7) : DIM D(12) : DIM N(12)
  60    LET B$ = "   "
  65    LET R$="----------------------------"
  70    INPUT "Entre el ano: ",A$
  80    LET Y = VAL(RIGHT$(A$,2)) : LET C = VAL(LEFT$(A$,2))
  90    LET A = VAL(A$)
 100       GOSUB 1000 : GOSUB 3000
 110    FOR F = 1 TO 12
 120       GOSUB 4000
 130       LET CD = 1 : LET II = 1
 140       LET M = N(F)
 150       GOSUB 2000
 170       IF II>7   OR II>R THEN GOTO 210
 175          PRINT B$;
 180          LET II = II+1
 190          GOTO 170
 210       IF CD>D(F) THEN GOTO 320
 230          IF II>7   THEN PRINT : LET II = 1
 260          IF CD<10 THEN PRINT  " ";
 265          PRINT CD;
 280          LET CD = CD+1 : LET II = II+1
 310          GOTO 210
 320       PRINT : PRINT R$
 330       IF INKEY$="" THEN GOTO 330
 340       NEXT F
 350  END
1000 REM Nombres de los meses
1010    FOR I = 1 TO 12
1020       READ M$(I)
1030       NEXT I
1040 REM Nombres de los dias de la semana
1050    FOR I = 1 TO 7
1060       READ D$(I)
1070       NEXT I
```

```
1080 REM Codigos de los meses
1090    FOR I = 1 TO 12
1100       READ N(I)
1110       NEXT I
1120    RETURN
2000 REM Calculo del primer dia de cada mes
2010    LET D = 1
2020    IF M>10 THEN LET YN = Y-1 : GOTO 2030
2025    LET YN = Y
2030    LET A = INT((13*M-1)/5)
2040    LET B = INT(YN/4)
2050    LET Z = INT(C/4)
2060    LET X = A+B+Z+D+YN-2*C
2070    LET R = X-INT(X/7)*7
2080    IF R=0 THEN LET R = 6 : GOTO 2090
2085     LET R = R-1
2090    RETURN
3000 REM Numero de dias del mes de febrero
3010    LET R1 = A-INT(A/4)*4
3020    LET R2 = A-INT(A/100)*100
3030    LET R3 = A-INT(A/400)*400
3040    LET D(2) = 28
3050    IF R1=0 AND R2<>0 AND R3<>0 THEN LET D(2) = 29
3060 REM Numero de dias de los restantes meses
3070    LET D(1) = 31
3080    FOR I = 3 TO 12
3090       READ D(I)
3100       NEXT I
3110    RETURN
4000 REM Impresion de la cabecera
4010    CLS
4020    PRINT TAB(7);M$(F)
4030    PRINT R$
4040    FOR J = 1 TO 7
4050       PRINT D$(J);
4060       NEXT J
4070    PRINT : PRINT R$
4080    RETURN
5000 REM Almacen de datos
5010    DATA "ENERO","FEBRERO"
5020    DATA "MARZO","ABRIL","MAYO","JUNIO"
5030    DATA "JULIO","AGOSTO","SEPTIEMBRE"
5040    DATA "OCTUBRE","NOVIEMBRE","DICIEMBRE"
5050    DATA "  LU","  MA","  MI","  JU","  VI","  SA","  DO"
5060    DATA 11,12,1,2,3,4,5,6,7,8,9,10
5070    DATA 31,30,31,30,31,31,30,31,30,31
```

Veamos el funcionamiento del programa. En primer lugar dimensionamos varias variables (línea 50). Estas variables contendrán:

- M$: Lista de los nombres de cada mes.
- D$: Lista de nombres de los días de la semana.
- D: Lista de número de días de cada mes.
- N: Lista de código numérico de cada mes, para el algoritmo de Zeller.

Asignamos a continuación dos variables; la variable B$ nos servirá para escribir espacios en blanco en las zonas correspondientes del calendario, y R$ como línea de separación.

Procedemos entonces (línea 70) a introducir el año, y a efectuar los cálculos de las variables que se utilizan en las rutinas del algoritmo de Zeller y de los años bisiestos (líneas 80 y 90).

Se procede a continuación a leer los nombres de los meses, días y códigos (rutina 1.000), así como a leer o calcular el número de días de cada mes (rutina 3.000).

Ahora podemos ya iniciar el proceso repetitivo, el bucle de escritura de cada mes. En primer lugar se imprimirá la cabecera (rutina 4.000). A continuación se prepararán dos contadores; el primero es CD –contador de los días– y contendrá el valor del días que vamos a escribir. El segundo es II, e indica la posición que se ha escrito. Dentro de cada fila hay que escribir 7 posiciones, ya sean espacios en blanco (si estamos en la primera línea) o números. Cada vez que avancemos una posición, incrementaremos este contador.

Vamos ahora a calcular la posición del primer día del mes correspondiente. Para ello asignamos a la variable *M* el código del mes que corresponda (línea 140), y efectuamos los cálculos en la rutina 2.000. Esta rutina nos devolverá una variable –*R*–, cuyo valor indicará la posición para escribir el primer número. (Observe que en este sentido hacemos un cambio con respecto a lo que habíamos visto en el programa correspondiente; aquí modificamos el resultado de la forma más adecuada a nuestros fines (línea 2.080), con lo que la variable *R* nos indicará cuántos espacios en blanco hay que dejar antes de escribir el primer número).

Una vez conocido el valor de R, escribiremos los espacios correspondientes tal como vemos en las líneas 170 a 190. Para escribir los espacios en blanco, utilizaremos la variable *B$*, definida al principio del programa. Para que el cursor no salte de línea acabamos la instrucción con un punto y coma (línea 175). Para cada espacio escrito incrementaremos el contador de posición II. Cuando este contador alcance el valor de *R*, finalizamos el proceso, y empezaremos a escribir los números. Esto lo hacemos entre las líneas 210 a 310. En ellas vamos escribiendo números e incrementando dos contadores: el contador de días *(CD)* y el contador de posición *(II)*. Cuando el contador de posición es mayor que 7, indica que hemos finalizado una línea, saltamos a la inferior y volvemos a inicializar este contador (línea 230). El contador de día también lo vamos incrementando para cada día que escribimos, hasta haber escrito todos los de cada mes. En este momento CD tendrá un valor superior al elemento correspondiente de la lista D, que tal como hemos dicho contiene el número de días de cada mes, con lo que saldremos fuera de este bucle, para pasar a escribir el siguiente mes, hasta haber completado el calendario.

```
2070   LET R = X-INT(X/7)*7
```

Tal como hemos dicho, si su ordenador posee la función MOD, podremos substituir la línea 2.070 por:

RESUMEN

Existen diversos algoritmos para buscar un elemento dentro de una lista.

Un procedimiento para ello consiste en recorrerla elemento a elemento desde el principio. Este procedimiento de búsqueda se denomina búsqueda secuencial. El proceso termina cuando se ha encontrado el elemento o cuando se ha llegado al final de la lista.

Existen sistemas mejorados de búsqueda, que permiten alcanzar el resultado más rápidamente.

Uno de estos procedimientos es el algoritmo de búsqueda binaria. Para poder aplicar este algoritmo es necesario que la lista esté ordenada.

Para realizar una búsqueda binaria sobre una lista se parte por la mitad y se compara el valor buscado con el elemento intermedio. Si es mayor, se toma la mitad derecha de la lista (suponiéndola ordenada de menor a mayor), si es menor se toma la mitad izquierda.

Este procedimiento se va repitiendo, efectuando sucesivas particiones de la lista, hasta encontrar el elemento que se busca o bien hasta que no nos quede ningún elemento en el lista. En este caso podremos afirmar que el elemento que buscamos no está en la lista.

Otro algoritmo de interés en el manejo de listas es el procedimiento para mezclar o fusionar listas, conocido como algoritmo de Simple Merge.

Este algoritmo consiste en construir una lista única partiendo de otras dos, que deben estar ordenadas.

El procedimiento se basa en ir tomando el elemento mayor (supuesto que las listas están ordenadas de mayor a menor) entre las dos listas, escribirlo en la resultante y tacharlo de la que corresponda. Se comparan entonces los elementos restantes de ambas, y se va repitiendo el proceso hasta que en una de las listas no quede ningún elemento por tachar. Los restantes elementos de la otra se podrán escribir directamente en la lista resultante.

EJERCICIOS DE AUTOCOMPROBACIÓN

Completar las frases siguientes:

21. Un algoritmo que realiza una búsqueda dentro de una lista avanzando elemento a elemento del principio al fin se denomina algoritmo de búsqueda ....

22. El algoritmo de búsqueda secuencial puede aplicarse a .... lista.

23. El algoritmo de búsqueda binaria se puede aplicar únicamente a listas ....

24. El algoritmo que va efectuando sucesivas particiones de la lista para buscar en ella un elemento dado se denomina algoritmo de búsqueda ....

25. El procedimiento de .... sirve para mezclar o fusionar dos listas.

26. Para poder aplicar el algoritmo de Simple Merge es necesario que las listas estén ....

27. El algoritmo de Simple Merge finaliza cuando .... las dos listas.

28. Para confeccionar un calendario aplicamos el algoritmo de Zeller, para saber .... cae el 1 de enero.

29. Para confeccionar un calendario hay que saber además si el año es o no ....

30. La función .... nos permite calcular directamente el resto de una división.

Encierre en un círculo la letra que corresponde a la alternativa correcta:

31. El algoritmo de búsqueda secuencial sirve para:

a) Mezclar listas.

b) Ordenar listas.

c) Buscar un elemento dentro de una lista.

d) Confeccionar un calendario.

32. El algoritmo de búsqueda secuencial se basa en:

    a) Comparar el elemento buscado con todos los de la lista, del principio al fin.

    b) Comparar los elementos de la lista uno a uno entre sí.

    c) Recorrer la lista del principio al final.

    d) Incluir el elemento buscado en la lista.

33. Si el elemento no está en la lista el algoritmo de búsqueda secuencial debe:

    a) Indicarlo y finalizar.

    b) Seguir hasta que se incluya.

    c) Esperar que el usuario decida qué hacer.

    d) No tomar ninguna decisión.

34. El algoritmo de búsqueda binaria se puede aplicar sobre listas:

    a) De longitud infinita.

    b) Sólo de números.

    c) Sólo de textos.

    d) Ordenadas.

35. El algoritmo de búsqueda binaria va efectuando:

    a) Particiones sucesivas de la lista.

    b) Sumas sobre los elementos de la lista.

    c) Divide por dos los elementos de la lista.

    d) Multiplica por dos los elementos de la lista.

36. Si el elemento buscado no está en la lista, el algoritmo de búsqueda binaria:

    a) Finaliza cuando no puede seguir dividiendo la lista.

    b) Vuelve a empezar por el principio.

    c) No acaba nunca.

    d) Incluye el elemento que busca, dando su posición.

37. El algoritmo Simple Merge sirve para:

a) Generar listas muy grandes.

b) Partir listas grandes en otras más pequeñas.

c) Ordenar listas.

d) Fusionar o mezclar listas.

38. El algoritmo de Simple Merge finaliza cuando:

a) Los elementos de las dos listas iniciales se han pasado a la lista resultante.

b) No queda ningún elemento en la primera lista.

c) No queda ningún elemento en al segunda lista.

d) La lista resultante está vacía.

39. Para poder aplicar el algoritmo de Simple Merge, las dos listas deben estar:

a) Vacías.

b) Ordenadas.

c) Llenas.

d) Escritas.

40. La función MOD (X, Y)f equivale a:

a) Calcular el resto de la división de X por Y.

b) Calcular si el año Y es bisiesto.

c) Calcular el cociente de la división X por Y.

d) Determinar si X e Y son correctas.



Capítulo 17
• Introducción al estudio de fichero y al lenguaje máquina

ESQUEMA DE CONTENIDO

- Objetivos
- Ficheros de datos
  - Estructura de la información
    - Concepto de campo
    - Concepto de registro
    - Concepto de fichero
  - Directorio
- Tipos de fichero
  - Secuenciales
  - Directos
  - Indexados
- Acceso a un fichero
  - Apertura
  - Acceso
  - Cierre
- Iniciación al lenguaje máquina
  - Descripción del ordenador
  - Sistema hexadecimal
  - Organización interna de la CPU
    - Unidad de control
    - Reloj
    - Unidad aritmético-lógica
    - Registros
    - Memoria principal
    - Canales de comunicación
    - Concepto de microprocesador
  - Código máquina
  - Instrucción PEEK
  - Instrucción POKE
  - Función USR
- Sistemas operativos

17.0 OBJETIVOS

Hasta el capítulo 16, hemos aprendido la síntaxis de las instrucciones de BASIC y también hemos aprendido a utilizar las técnicas de programación, a fin de elaborar y poner a punto un programa.

Sin embargo, hay una serie de temas sobre informática, que no siempre están directamente relacionados con la programación pero que tienen gran importancia si se quiere dominar el manejo de un ordenador. A ellos vamos a dedicar estos dos capítulos que nos restan para finalizar la Enciclopedia.

Este primero, dedicado a lo que podríamos llamar temas avanzados, tiene el objetivo de acercarle un poco más al conocimiento interno de la máquina, así como a algunos temas de interés no directamente relacionados con el lenguaje de programación, pero también necesarios para comprender mejor el funcionamiento y las posibilidades que le ofrece su ordenador.

En primer lugar se estudiarán los distintos sistemas para almacenar información, así como la organización de la misma. Se definirán en este sentido los conceptos de directorio, fichero, registro y campo, que constituyen los distintos niveles en la estructura jerárquica de la información.

Se estudiará además la disposición física real de esta información, con lo que se verá el concepto de sector o bloque, también conocido como registro físico.

Según la organización de los registros en los ficheros se estudiarán los distintos tipos en que estos se clasifican: secuenciales, directos e indexados, así como la forma de acceder a ellos (abrir y cerrar, leer y escribir) desde un punto de vista general, ya que la sintaxis concreta de estas operaciones varía según el modelo de ordenador.

Por otra parte, intentaremos iniciarle en el concepto de lenguaje máquina, lo que nos llevará a ver más de cerca el funcionamiento interno del ordenador y la forma de trabajar de cada componente: unidad de control, reloj, unidad aritmética-lógica, registros, memoria, etc. En este sentido aprenderemos dos nuevas instrucciones de Basic (PEEK y POKE) que nos permiten acceder directamente a una posición determinada de memoria, así como la función USR, que nos permite ejecutar programas en lenguaje máquina.

Finalmente se verá de forma general el concepto de Sistema operativo, así como las diversas funciones y posibilidades que éstos ofrecen.

17.1 FICHEROS DE DATOS

En muchos casos, los programas necesitan operar con datos cuya vigencia es continua. Tales datos serán utilizados transcurrido un cierto período de tiempo por el mismo o por otros programas. Ejemplos de este tipo de datos podrían ser una lista de artículos con sus precios o una relación de personas con sus direcciones y números de teléfono.

La rentabilidad del ordenador se muestra al procesar gran capital de datos

Por otra parte, ya sabemos que la utilización de un ordenador será más rentable que el proceso manual cuando un proceso se efectúe repetidas veces sobre una gran cantidad de datos. Por tanto, la información

a la que nos estamos refiriendo tiene dos características importantes: es de tipo masivo (gran cantidad de datos) y tiene una vigencia que supera el tiempo de una sesión de trabajo.

Dispositivos de almacenamiento

Para almacenar esta información será necesario disponer de dispositivos de memoria auxiliar, generalmente de tipo magnético, los cuales tienen una capacidad bastante mayor que la memoria principal del ordenador. Los más utilizados son *cassettes y diskettes* para microordenadores y *discos* y *cintas* para ordenadores mayores. Si queremos que no decaiga la rapidez de proceso del ordenador, la información almacenada en estos dispositivos estará en formato directamente legible por el ordenador. De este modo, se evitan las conversiones de formato que implicarían pérdidas de tiempo. Recordemos que el resto de unidades periféricas sí realizan esta conversión. Así, por ejemplo, una impresora cambia los datos con formato interno (utilizable por el ordenador) a datos con formato externo (utilizable por el hombre).

Los datos almacenados deben estar organizados

Además de los dispositivos de memoria auxiliar, es necesario un último requisito. Para que el tratamiento de la información se haga de forma eficiente, deberemos almacenar los datos con una organización adecuada. Un conjunto de datos almacenados en un dispositivo de memoria auxiliar y dispuestos con una estructura u organización determinada, recibe el nombre de *fichero* o *archivo.*

En resumen, las cuatro características fundamentales de un fichero son:

1. Contienen un número elevado de datos.
2. La información tiene una vigencia superior al de la ejecución de un proceso.
3. Se almacena en dispositivos de memoria auxiliar.
4. En las transferencias ordenador-memoria auxiliar no hay conversión de formato.

17.1.1 Estructura de la información

Estructura jerárquica de la información

Para comprender la organización de un fichero, primero hay que conocer la estructura de la información. Esta estructura es de tipo jerárquico y parte de una unidad que es el carácter o su equivalente en unidad de memoria que es el byte. Aunque, de hecho, la unidad más pequeña es, como sabemos, el bit, este se emplea pocas veces como elemento separado.

El esquema de la estructura de la información se representa en la figura 1. En los apartados que siguen a continuación daremos una explicación más detallada sobre cada uno de los términos.

17.1.1.1 Concepto de campo

Para facilitar las explicaciones de estos conceptos, haremos unas analogías con los sistemas manuales de almacenamiento de la información. Supongamos que tenemos un archivo de clientes. los datos de cada

Figura 1. Esquema de la estructura de la información.

```
                                            { n CARACTERES
                               { CAMPO 1  {       ó
                 { REGISTRO 1  { CAMPO 2  {     BYTES
                 { REGISTRO 2  {    •
    FICHERO      {    •        { CAMPO n
                 {    •
                 {    •
                 { REGISTRO n
```

cliente los tenemos escritos sobre una ficha de papel o cartulina. Recogeremos cuatro datos por cliente, que serán: nombre, dirección, condiciones de pago y saldo.

> **Un campo es una *sucesión de bytes* o carácteres que contiene una *información determinada.***

Definición de Campo

Equivale a cada uno de los apartados de una ficha. En el ejemplo, los campos serían el nombre, la dirección, las condiciones de pago, etc. Cuando se define un campo hay que establecer dos características. La primera, es su longitud o, dicho en otras palabras, el número de caracteres que se destinan para dicho campo. La segunda, es el tipo de dato que se almacena, es decir, si se trata de un dato textual o numérico. De hecho, un campo es en muchos aspectos, equivalente a una variable y por tanto comparte varias de sus características: almacena un dato único y requiere que se indique el tipo de dato que contiene.

17.1.1.2 Concepto de registro

> **Un registro es un conjunto de campos que forman una unidad *lógica.***

Definición de Registro

Equivale al concepto de ficha en un archivo manual. Por tanto, en el ejemplo anterior, el conjunto de datos referido a un cliente (contenidos en la ficha) constituyen un registro. Además, hemos dicho que el conjunto de datos forma una unidad lógica. ¿Qué significa esta afirmación? Los campos considerados individualmente no pueden agruparse de cualquier forma. Si en el ejemplo anterior hubiésemos agrupado todos los nombres formando un sólo registro, todas las direcciones formando otro registro, etc. está claro que el procesado de la información se hubiera visto dificul-

tado enormemente. La causa de esta ineficiencia es que los registros no son homogéneos, puesto que cada uno contiene registros deben agrupar campos que en conjunto forman una unidad lógica. En el ejemplo citado, esta idea es clara, pero hay que reconocer que en ficheros más complicados no es tan evidente el concepto de unidad lógica.

No existe una equivalencia exacta en BASIC para el concepto de registro. En principio, podría pensarse que un conjunto dimensionado (una lista o un vector) es equivalente a un registro. De la misma forma que un registro está formado por varios campos, un conjunto dimensionado contiene varios datos. Sin embargo, existe una diferencia fundamental. Como estudiamos en su momento, un conjunto dimensionado es homogéneo y por tanto, todos los datos que contiene son exactamente del mismo tipo, o todos textuales a todos numéricos. Por el contrario, un registro puede agrupar campos de distinto tipo. En el ejemplo, cada registro contenía campos textuales (hombre y dirección) y campos numéricos (condiciones de pago y saldo).

17.1.1.3 Concepto de fichero

Finalmente, un fichero o archivo será una colección de información relacionada lógicamente, agrupada en registros.

Definición de Fichero

Siguiendo con nuestro ejemplo, el conjunto de fichas de clientes constituyen, obviamente, el fichero. En la figura 2 se muestra la estructura de este fichero, mientras que en la figura 3 podemos ver la equivalencia de los conceptos estudiados con los de un fichero manual.

Como de costumbre, el tamaño de un registro se medirá en bytes o caracteres. Este tamaño será igual a la suma de longitudes de los campos que lo componen. Lógicamente, el tamaño del fichero se obtendrá multiplicando el tamaño del registro por el número total de registros que contiene el fichero.

Cuando se diseña un programa que maneje ficheros, una de las primeras operaciones que se realizan es evaluar su tamaño total. Para ello, en primer lugar, se asignan unos valores provisionales a las longitu-

Figura 2. Estructura de los campos y registros de un fichero de clientes.

FICHERO									
REGISTRO-1				REGISTRO-2				R	
CAMPO 1	CAMPO 2	CAMPO 3	CAMPO 4	CAMPO 1	CAMPO 2	CAMPO 3	CAMPO 4	CAMPO 1	CA
NOMBRE	DIRECCION	COND. PAGO	SALDO	NOMBRE	DIRECCION	COND. PAGO	SALDO	NOMBRE	DI

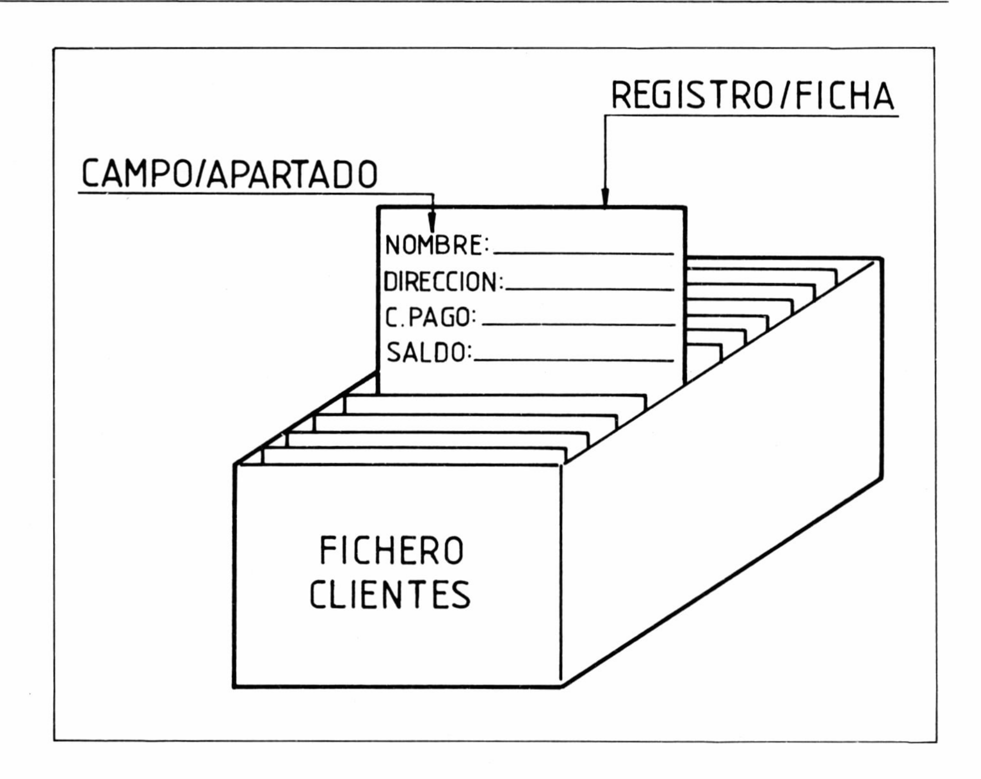


Figura 3. Equivalencia de los conceptos de registro/ficha y campo/apartado con un fichero manual.

des de los campos. Siguiendo con nuestro fichero de clientes como ejemplo, podríamos realizar la siguiente distribución:

Campo	**Longitud**
Nombre	30 by.
Dirección	50 by.
Cond. pago	3 by.
Saldo	5 by.
Total	88 by.

Como vemos, la longitud de cada registro resulta ser de 88 bytes. Por tanto, si tenemos un total de 300 clientes, el fichero tendrá un tamaño total de 88 × 300 = 26.400 by o, hablando en términos informáticos, de aproximadamente 26 Kby. Podría darse el caso de que en nuestro dispositivo de memoria auxiliar no dispusiésemos de esta cantidad de memoria, sino de algo menos, por ejemplo 23 Kby. Entonces se realiza una nueva asignación de tamaños, reduciendo alguno de los campos. Si asignamos 40 by al campo Dirección, en lugar de 50, el registro tendrá entonces 78 by lo que conduce a un tamaño de fichero de aproximadamente 23 Kby.

17.1.2 Directorio

Definición de Directorio

Dado que un dispositivo de memoria auxiliar tiene una capacidad muy elevada, normalmente contiene más de un fichero. Los ficheros se

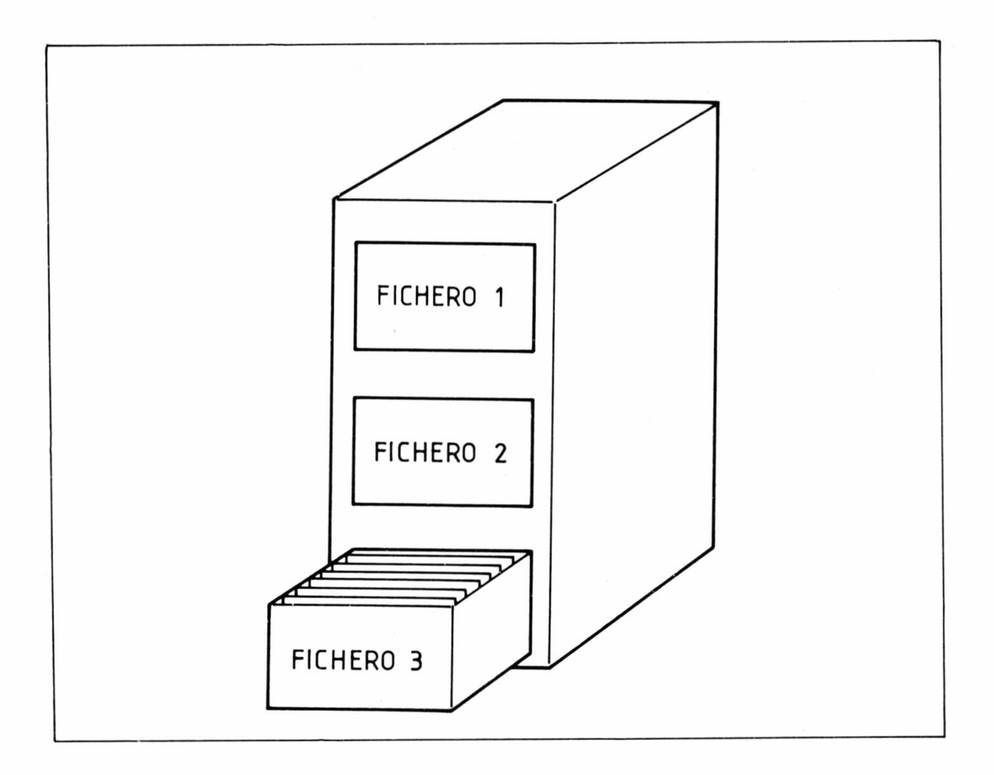


Figura 4. Archivador que contiene varios ficheros. Es equivalente al concepto informático de Directorio.

suelen agrupar formando lo que se denomina *directorio*. Por tanto, *un directorio equivale a un archivo de ficheros,* tal como se ve en la figura 4.

El directorio es un índice o catálogo de ficheros que facilita al ordenador la localización de los datos en un dispositivo de almacenamiento. Llegados a este punto hay que explicar algunos conceptos fundamentales sobre cómo se almacena realmente la información. Los dispositivos de memoria auxiliar tienen una capacidad que varía desde algunos cientos de miles de bytes hasta varios cientos de millones de bytes. Ciertamente, para microordenadores se emplean dispositivos pequeños con capacidades no muy elevadas. Sin embargo, independientemente de su tamaño, en todos ellos los bytes se agrupan formando lo que se denomina *sector* o *bloque*. El tamaño de este sector o bloque viene impuesto por el diseño electrónico del dispositivo, es decir, es algo que decide el fabricante y en el cual no podemos intervenir. Valores usuales de un sector o bloque son 256, 512 o 1.024 bytes.

Registro físico y registro lógico

Observemos ahora un detalle importante: un fichero con varios miles de bytes tendrá simultáneamente dos tipos de subdivisiones. Por una parte, estará subdividido en registros y por otra, en sectores o bloques. La primera subdivisión es de tipo lógico, puesto que la hemos creado nosotros y podemos cambiarla cuando nos interese. La segunda es de tipo físico y depende exclusivamente de la máquina. Por esta razón, un sector o bloque se denomina a veces *registro físico* y al registro normal se le denomina *registro lógico.*

Para comprender mejor estos conceptos haremos una analogía con un libro. Un libro también está formado por muchos miles (a veces millones) de caracteres. El autor ha hecho una agrupación de estos caracteres en capítulos, mientras que el impresor ha realizado una agrupación de

caracteres en páginas. Los capítulos son el equivalente de un registro, puesto que reflejan una separación lógica entre cada grupo de caracteres. Por el contrario, una página es equivalente a un sector puesto que se realiza una separación de los caracteres según el formato del papel sin tener en cuenta el contenido del texto.

Observemos otro detalle importante: al abrir un libro, lo hacemos siempre por una página y no por un capítulo (excepto que coincidan por casualidad). Aquí aparece el problema. Por una parte nosotros queremos localizar un capítulo determinado y por otra, sólo podemos localizar páginas aisladas. Para facilitar esta tarea, existe el índice que establece la relación entre capítulos y páginas. Pues bien, un directorio cumple exactamente esta misma misión entre ficheros y sectores.

Un directorio o catálogo es una tabla que contiene el nombre del fichero, su localización en el dispositivo y normalmente, también el tamaño del fichero (Fig. 5a). Además, a veces, se suele guardar más información en la tabla, como por ejemplo, la fecha de creación, nombre del usuario, etc. Esta información adicional varía mucho de una máquina a otra.

Cuando un programa va a trabajar con un fichero determinado, encuentra de forma inmediata en el directorio el número de sector donde empieza dicho fichero (ver figuras 5a y 5b), sin necesidad de realizar una búsqueda exhaustiva, lo que conllevaría una gran pérdida de tiempo. Hay que advertir que estas operaciones son internas y totalmente invisibles para el usuario.

La mayoría de ordenadores, aunque trabajan siempre con sectores físicos, permiten que el programador trabaje directamente mediante registros lógicos, facilitando así la elaboración de un programa.

17.2 TIPOS DE FICHEROS

Clases de ficheros

Según la organización de los registros, existen tres tipos básicos de ficheros: *Secuenciales, Directos* e *Indexados*. Dentro de cada grupo existen algunas variantes que se diferencian ligeramente entre sí. Hay que tener en cuenta que el soporte físico del fichero condiciona, a veces, su organización. Así, una cinta magnética o un cassette, sólo puede contener ficheros secuenciales, mientras que un disco o un diskette admite los tres tipos.

17.2.1 Secuenciales

Secuenciales

Se llaman ficheros *secuenciales* cuando los registros se encuentran dispuestos de forma sucesiva (secuencial). Entre cada registro hay una separación formada por uno o varios bytes (Fig. 6). Un carácter muy usado como separación es el carácter 13 del código ASCII que equivale a la tecla de fin de línea. Al existir esta marca de separación, los registros pueden ser de longitud variable. Esto trae como ventaja que la información se guarda de forma muy compacta. La característica principal de estos ficheros es que el acceso a los registros es siempre consecutivo;

Figura 5. a) Tabla de datos que suele contener un directorio informático. b) Disposición de los ficheros sobre los sectores o bloques de un dispositivo de memoria auxiliar.

DIRECTORIO		
NOMBRE	LOCALIC.	TAMAÑO
FICHERO 1	0	7
FICHERO 2	7	4
…	…	…
FICHERO n	100	6

a)

SECTORES ó BLOQUES

0 1 2 3 4 5 6 7 8 9 10 11 105 106

FICHERO 1

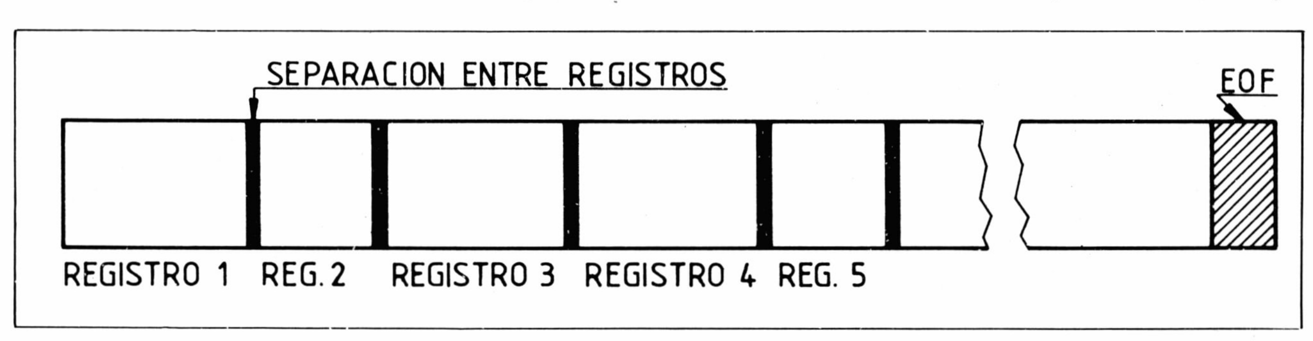
FICHERO 2

b)

es decir, que para acceder a un registro determinado hay que pasar previamente por todos los que le preceden. Cuando se accede a un registro (para leer o grabar) se entiende siempre que la operación se efectúa sobre el registro siguiente al último utilizado.

Una buena analogía con un fichero secuencial es un cassette de audio. Si grabamos varias canciones en un cassette, probablemente deja-

Figura 6. Esquema de un fichero con organización secuencial.



remos entre ellas un espacio sin grabar. Esta separación equivale a la separación entre registros. Por otra parte, para escuchar una canción situada hacia el final deberemos pasar previamente por encima de las canciones que le preceden. Si bien en un cassette de audio existe la operación de avance rápido, no es posible efectuarla en informática ya que el avance es siempre a la misma velocidad.

Aparte de la lectura y la grabación, sobre un fichero secuencial se pueden realizar otras operaciones que son:

a) Rebobinar o colocarse otra vez a inicio del fichero (sobre el primer registro).

c) Añadir al final. Se utiliza para alargar el fichero, o sea, para añadir registros nuevos detrás de los registros ya existentes. A veces, la única posibilidad consiste en crear un fichero nuevo a partir del original.

d) Retroceso. Significa retroceder un registro. No siempre es posible esta operación.

Los ficheros secuenciales se caracterizan por ser muy compactos y porque el acceso a un registro es muy laborioso. la primera característica hace que se utilicen para almacenar información OFF-LINE (no accesible inmediatamente por el ordenador) como por ejemplo copias de seguridad y archivos históricos. La segunda característica sólo permite que se utilicen en las tareas normales, cuando precisamente se vayan a procesar los registros correlativamente, como por ejemplo los movimientos de entrada-salida de un almacén, apuntes contables, etc. En este caso se trata de datos efímeros, que una vez procesados carecen de interés, excepto como información histórica.

Detrás del último registro se dispone una agrupación típica de bits llamada EOF que es la abreviatura de End Of File (Fin de fichero, en inglés) que nos delimita el final del fichero (Fig. 6).

17.2.2 Directos

Directos

Se denominan también, a veces, *relativos* o de *acceso al azar*. La longitud del registro es fija y no existen, por tanto, marcas de separación entre ellos (Fig. 7a). Si la longitud del registro es 88, por ejemplo, sabemos que los bytes que van desde el 1 hasta el 88 pertenecen al primer registro, del 89 hasta el 176 pertenecen al segundo y así sucesivamente. En consecuencia, es innecesario colocar bytes de separación. Por otra parte, la longitud fija permite que la posición de un registro determinado se pueda conocer mediante un simple cálculo, que consistirá en la multiplicación de la longitud por el número de registro. Siguiendo con el ejemplo, el registro tercero terminará en el byte 88 x 3 = 264 y por tanto ocupará desde el 177 al 264. De nuevo hay que señalar que estas operaciones las realiza internamente el ordenador, mientras que el programa se limita a darle el número de registro. Por tanto, en un fichero directo se accede a un registro conociendo simplemente su número de orden relativo.

Aunque sobre el soporte físico (diskette o disco) el fichero se encuentre grabado tal como se ve en la figura 7a, para nosotros será mucho

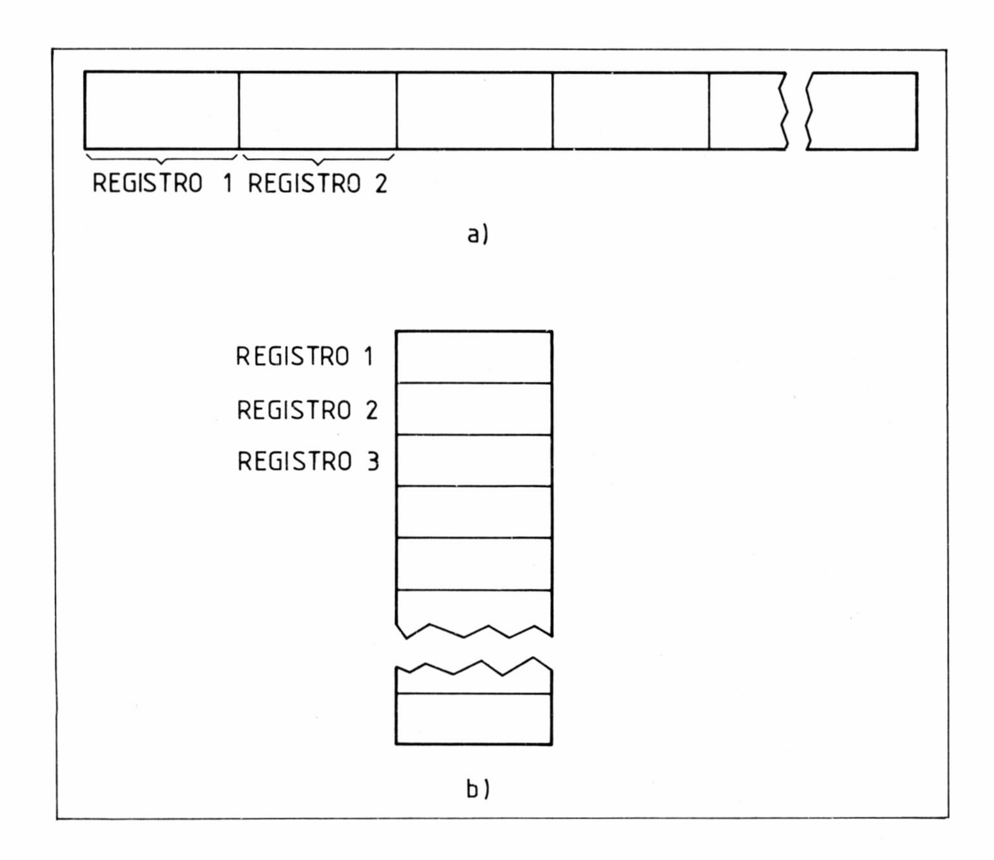


Figura 7. a) Esquema lineal de un fichero con organización directa o relativa. b) Esquema tabular del mismo fichero.

más útil emplear una representación tabular como se muestra en la figura 7b.

En estos ficheros sólo están definidas las operaciones de acceso (lectura y grabación) ya que carecen de sentido las operaciones de rebobinado, retroceso, etc. puesto que se pueda acceder a cualquier registro en cualquier momento.

Se usan normalmente para almacenar datos de vigencia continua (aunque modificables), como por ejemplo, un fichero de clientes o de productos. Son los ficheros de acceso más rápido, pero tienen el inconveniente de que para acceder a una ficha (registro) determinada es necesario conocer su número. En nuestro fichero de clientes, para localizar un cliente determinado será necesario conocer el número de registro en que se encuentra. En estos casos se suele disponer de unos listados impresos que contienen una relación de los nombres con los números de registro asociados.

Otro inconveniente más grave aparece cuando se desea dar de baja una ficha. En este caso se suele colocar una marca de anulado, puesto que normalmente resulta inviable trasladar una posición hacia atrás a todos los registros situados a continuación.

17.2.3 Indexados

Indexados

Estos ficheros están diseñados para superar los inconvenientes de los ficheros directos. Existe un gran número de varintes de este tipo de

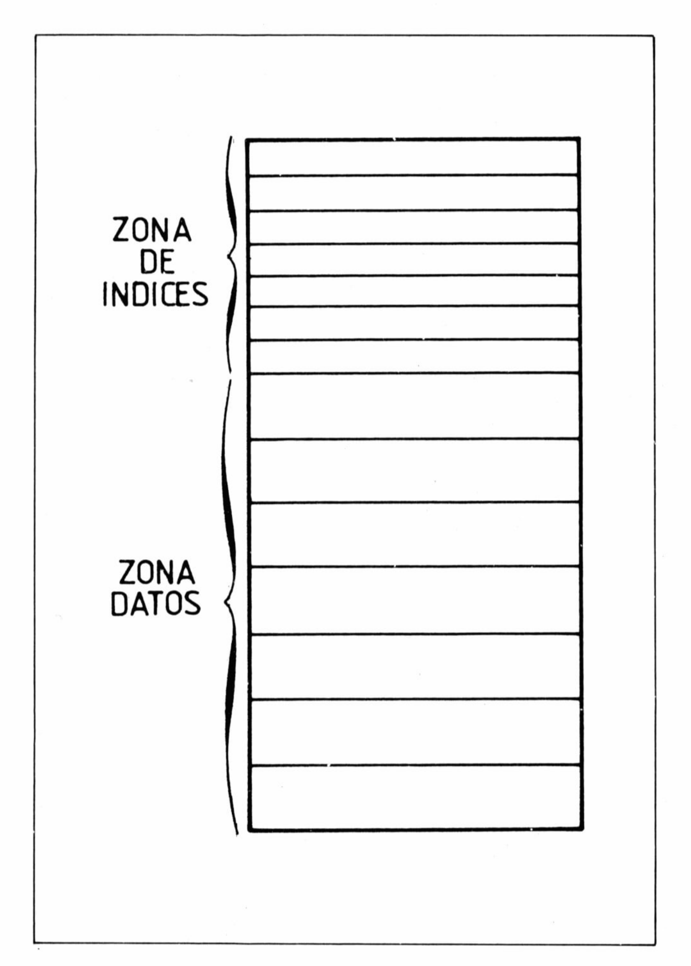


Figura 8. Esquema de un fichero indexado.

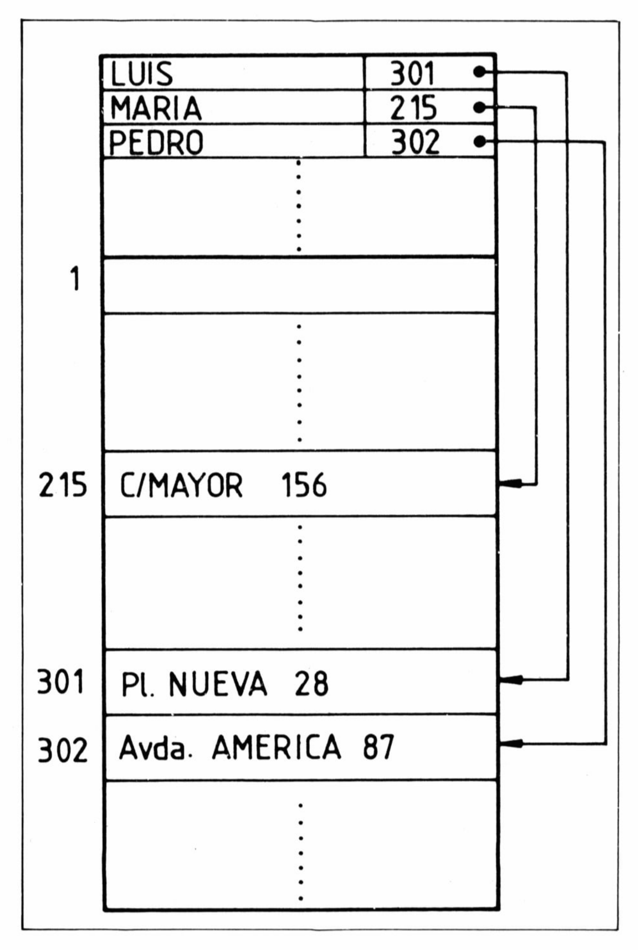


Figura 9. Asociación entre claves de acceso y número de registro en un fichero indexado.

ficheros y, por tanto, sólo se mencionarán las características generales.

Los registros son de longitud fija, al igual que en los ficheros directos, pero no se accede a ellos mediante un número, sino que en su lugar se utiliza una serie de caracteres que constituyen la llamada «clave» de acceso. Esta clave puede ser un código, un nombre, etc. y es la única forma de acceder al registro asociado.

Para poder realizar esta operación, un fichero indexado se encuentra dividido en dos zonas como se ve en la figura 8. Una de las zonas contiene los registros con todos los datos, como si se tratara de un fichero directo. Los registros de la otra zona, llamada zona de índices, son mucho más pequeños y contienen la clave de acceso y el número de registro donde se encuentran los datos asociados a dicha clave. En la figura 9 se muestra un ejemplo de fichero indexado que contiene una relación de nombres y direcciones. Las claves de acceso son los nombres. De esta forma, cuando se solicita una clave, se efectúa una búsqueda sobre la

zona de índices cuyo tamaño es muy inferior a la zona de datos. Una vez localizada la clave, se accede directamente al registro de datos. Además, las operaciones de dar de alta o de baja son bastante rápidas, pues sólo implican la reorganización de la zona de índices.

Algunos ficheros indexados permiten solamente el acceso por clave. Otros, llamados ISAM («Indexed Sequential Access Method» o método de acceso secuencial-indexado) mantienen además el orden alfabético por claves. Finalmente, algunos tipos permiten, incluso, la existencia de más de una clave de acceso por registro.

Las operaciones que se pueden realizar sobre los ficheros indexados son:

a) Lectura de una clave existente. A diferencia de los ficheros directos, donde cada número equivalía a un registro, aquí una clave puede haber sido dado de alta o ser todavía inexistente.

b) Dar de alta o modificar. Si se graba una clave inexistente, se da de alta. Si ya existía, se actualizan los datos del registro correspondiente.

c) Dar de baja. Cuando se da de baja una clave existente, se borra de los índices y el registro pasa a la situación de disponible para insertar datos de una nueva clave.

d) Leer siguiente o anterior. Significa leer la clave siguiente o anterior según el orden alfabético. Lógicamente estas dos operaciones sólo son posibles en ficheros de tipo ISAM.

17.3 ACCESO A UN FICHERO

Operaciones en el manejo de la información

Hay tres operaciones fundamentales para manejar la información de un fichero, que son, *Apertura, Acceso* y *Cierre.*

Los ficheros se identifican mediante un nombre, de forma parecida a la nomenclatura de las variables de un programa. Por ejemplo, un fichero de clientes podría llamarse «CL» o bien «CLIENT», etc. Las reglas de nomenclatura varían de un ordenador a otro, aunque la longitud del nombre oscila usualmente entre 6 y 10 caracteres. A menudo, el nombre está formado por dos palabras separadas por un punto. La primera palabra es el nombre propiamente dicho y la segunda designa el tipo de información que contiene (si contiene datos, programas, etc.). Por ejemplo, «CLIENT.DAT» designaría un fichero llamado «CLIENT» que contiene datos. Sin embargo, insistimos, que las reglas de nomenclatura de un fichero no dependen del BASIC sino del ordenador (en realidad, del sistema operativo como veremos más adelante).

17.3.1 Apertura

La apertura informa al programa de cuál es el fichero sobre el que se va a trabajar. Para ello, se asigna un número (normalmente comprendido entre 1 y 16) a dicho nombre. A partir de entonces, el programa trabajará con dicho número. Un mismo programa puede tener varios ficheros

abiertos simultaneamente. A cada uno de ellos, en el momento de abrirlos, se le asignará un número distinto.

La instrucción para abrir un fichero tiene la forma general del tipo:

OPEN número, nombre

OPEN#

La sintaxis concreta depende de cada versión de BASIC. En las prácticas con el microordenador se indica la forma en que deberemos escribir esta instrucción en cuanto ordenador. Naturalmente, suponiendo que nuestro ordenador disponga de algún dispositivo de almacenamiento auxiliar (cassette, diskette,...).

La misión de la instrucción OPEN es doble. La primera consiste, como hemos mencionado, en asignar un número al fichero, de forma que no tengamos que repetir el nombre cada vez que lo utilizemos. La segunda misión es invisible y consiste en reservar una porción de la memoria central para que actúe de memoria intermedia durante el traspaso de la información del fichero a las variables del programa. Conviene recordar ahora lo que hemos explicado respecto al registro físico y registro lógico. La unidad de almacenamiento nos transmitirá siempre un registro físico. Por tanto, será necesario preparar una zona de memoria de tamaño de dicho registro para recibir la información. De aquí se extraerá la información de cada campo y se asignará a las variables correspondientes. Esta memoria intermedia es la que reserva la instrucción OPEN.

Frecuentemente, las memorias intermedias se designan por su nombre inglés, que es «buffer».

17.3.2 Acceso

Una vez abierto un fichero ya podemos acceder a sus registros. El acceso puede ser de lectura (obtener información del fichero) o de grabación (escribir algo en el fichero). Normalmente se trabaja siempre con un registro completo. A menudo hay que modificar información ya existente, para lo cual hay que realizar una lectura y una posterior grabación sobre el mismo registro. Esta operación recibe el nombre de acceso de actualización.

Para leer de un fichero se utiliza una versión modificada de la instrucción INPUT en la cual se indica el número de fichero. La sintaxis general es:

INPUT #n, lista variables

INPUT#

En donde *n* es el número de fichero asignado en el OPEN. El símbolo de sostenido (#) es muy frecuente en todos los lenguajes de programación (y no sólo en BASIC), como prefijo del número de fichero abierto. Además de esta instrucción, algunas variantes del BASIC utilizan también READ (leer) y GET (obtener). Todas ellas tienen en común de que leen un registro completo, asignando los contenidos de los campos a la lista de variables.

Hay que tener en cuenta que la ejecución de una instrucción de lectura no desencadena forzosamente una operación de acceso al dispositivo de almacenamiento. Aunque el usuario no nota la diferencia, generalmente se trata de dos procesos asíncronos (no simultáneos en el tiem-

po). Por ejemplo, supongamos que en un fichero el registro lógico tiene 64 bytes y que el tamaño de sector es 256 bytes. Entonces, cuando el usuario acceda al registro 5, la unidad nos transferirá al «buffer» un total de 256 / 64 = 4 registros lógicos; es decir, del registro 5 al 8. Si el usuario accede a continuación al registro 6 (o al 7 o al 8) no se producirá acceso al dispositivo de almacenamiento, puesto que ya están en la memoria intermedia. El programa obtendrá la información directamente de esta memoria. Sólo se producirá un acceso al dispositivo cuando el programa pida un registro que no se encuentre en memoria.

En la figura 10 se muestra un esquema del traspaso de la información utilizando la memoria intermedia creada por el OPEN.

Este proceso (que es automático, sin intervención del usuario) es idéntico para la grabación. De esta forma se agiliza la velocidad del proceso ya que los accesos a la unidad de almacenamiento se reducen al mínimo.

Para grabar un registro se utiliza la instrucción PRINT modificada. La forma general es:

PRINT #n, lista expresiones

PRINT#

donde *n* es el número de fichero. El resultado de las expresiones (que pueden ser simples variables) pasarán a ser el contenido de los campos. Además de PRINT, algunos BASICS utilizan también WRITE (escribir) y PUT (colocar o poner).

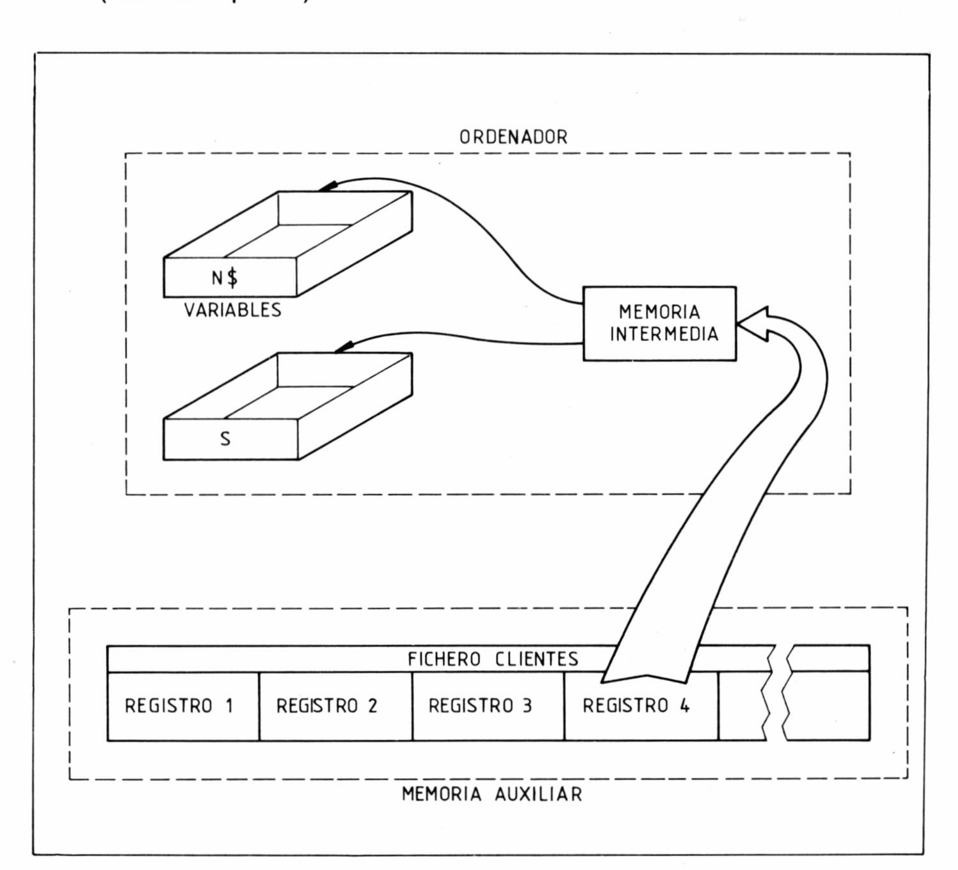


Figura 10. Traspaso de la información contenida en un fichero a las variables del programa. La memoria intermedia ha sido reservada por la instrucción OPEN.

Para actualizar un registro se utilizan combinadamente las instrucciones de lectura y grabación, si bien algunas variantes del BASIC tienen una instrucción especial denominada UPDATE (actualizar en inglés).

17.3.3 Cierre

Cuando se han terminado las operaciones de acceso hay que cerrar el fichero. Esta operación es importante por varias razones. En primer lugar, impide que a causa de una manipulación incorrecta se pueda dañar el fichero,. Una vez cerrado, las instrucciones de lectura o grabación no pueden operar sobre el fichero. En segundo lugar, libera la memoria intermedia («buffer») para que pueda ser utilizada para otras tareas. Además, en el caso de estar grabando registros, esta operación es especialmente importante. Recordemos que se trataba de un proceso asíncrono y que no se transfiere la información a la memoria auxiliar hasta que la memoria intermedia no está llena. Por tanto, es posible que al finalizar el proceso de grabación todavía permanezca en el «buffer» algún o algunos registros sin transferir. Al cerrar el fichero, se transfieren obligatoriamente y se recupera esta memoria intermedia.

Para cerrar un fichero se suele utilizar la instrucción CLOSE (cerrar en inglés) cuya síntaxis general es:

CLOSE #n

CLOSE#

en donde n es el número de fichero asignado en el OPEN.

RESUMEN

Para almacenar la información se utilizan dispositivos de memoria auxiliar, que nos permiten mantener cantidades de información de cualquier tamaño, durante el tiempo que sea necesario. Los dispositivos de memoria auxiliar para los microordenadores son cassettes o diskettes.

Para manipular eficientemente la información, ésta se organiza en ficheros o archivos. Estos archivos se dividen en registros, de la misma forma que un archivo manual se divide en fichas, y cada registro agrupa un conjunto de campos que constituyen una unidad lógica.

Un campo es una sucesión de bytes o caracteres que contienen una información determinada (cada uno de los apartados de la ficha en un archivo manual).

Dentro de los dispositivos de memoria auxiliar, los ficheros se pueden agrupar formando directorios. Estos equivalen pues a un archivo de ficheros.

Los dispositivos de memoria auxiliar tienen capacidad para almacenar gran número de bytes. Físicamente los bytes se agrupan en sectores o bloques, independientemente de la agrupación lógica (fichero, registro, campo), que posteriormente realice el usuario. El

tamaño de un sector o bloque (también conocido como registro físico) es fijo, y se determina en el momento de fabricar el dispositivo.

Según la disposición lógica de los registros en un fichero, éstos se dividen en:

Secuenciales: Disposición de los registros en forma sucesiva, separados por un determinado carácter. La información es muy compacta, y el acceso a un registro se hace en forma secuencial: para acceder a uno se deberá pasar por todos los anteriores.

Directos: Longitud fija de los registros, acceso directo a cada uno, mediante un pequeño cálculo; conociendo el número de registro localizaremos la información que contiene directamente.

Indexados: Registros también de longitud fija, acceso a través de una clave. En un fichero se encuentran dos clases de registros. Por una parte, registros de índices, que contienen la clave de acceso y la posición de los datos asociados a ella, y por otra, registros de datos, que contienen la información.

Las operaciones fundamentales para manejar la información de los ficheros son tres: apertura, acceso y cierre.

La apertura informa al programa de cuál es el fichero sobre el que se va a trabajar, y nos permite acceder a sus registros.

El acceso a un fichero puede ser de lectura (obtener información) o de grabación (escribir información).

Finalmente, cuando se han terminado las operaciones de acceso al fichero, éste se debe cerrar de forma que se impida el posterior acceso al mismo y se deje la información correctamente situada.

EJERCICIOS DE AUTOCOMPROBACIÓN

Encierre en un círculo la letra que corresponda a la alternativa correcta:

1. Los dispositivos de memoria auxiliar se utilizan cuando:

a) Se necesita una ampliación de la memoria principal.

b) El número de datos a almacenar es elevado.

c) Se necesita mayor capacidad de cálculo.

d) La información ha perdido vigencia.

2. La estructura de la información es de tipo:

   a) Jerárquico.
   b) Bit a bit.
   c) Byte a byte.
   d) Secuencial.

3. La información se estructura siguiendo el siguiente orden:

   a) Registro, Fichero, Byte, Bit, Campo.
   b) Bit, Campo, Registro, Byte, Fichero.
   c) Fichero, Registro, Campo, Byte, Bit.
   d) Campo, Registro, Fichero, Byte, Bit.

5. Un registro se define como:

   a) Un conjunto de campos que forman una unidad lógica.
   b) Un conjunto de ficheros.
   c) Un índice de ficheros.
   d) Un almacén de información.

4. Un campo se define como:

   a) Una sucesión de bytes o caracteres que contienen una información determinada.
   b) Un índice de ficheros.
   c) Un caso especial de fichero.
   d) Una agrupación de registro.

6. Un fichero o archivo se define como:

   a) Una colección de directorios.
   b) Un grupo de bytes.
   c) Una colección de información relacionada lógicamente, agrupada en registros.
   d) Un conjunto de archivos.

7. Un directorio se define como:

   a) Un índice o catálogo de ficheros.

   b) Un conjunto de registros.

   c) Un conjunto de Bytes.

   d) Un grupo de campos.

8. Los ficheros cuyos registro se encuentran dispuestos en forma sucesiva se denominan:

   a) Directos.

   b) Secuenciales.

   c) Indexados.

   d) Ficheros aunque no son ficheros.

9. Los ficheros directos se definen como aquellos:

   a) Cuyos registros se encuentran dispuestos en forma sucesiva.

   b) Ficheros con longitud fija de registro, de forma que se puede acceder directamente a cualquier registro, efectuando un simple cálculo.

   c) En los que se accede a los registros a través de una clave.

   d) Que tienen dos tipos de registros.

10. Las operaciones de apertura de un fichero sirve para:

   a) Vaciar la información que contiene.

   b) Impedir el acceso a los demás usuarios.

   c) Informar al programa de cuál es el fichero sobre el que se va a trabajar.

   d) Impedir el posterior acceso a la información.

11. La operación de cierre de un fichero sirve para:

   a) Proteger la información y vaciar la memoria intermedia.

   b) Mejorar la velocidad de acceso al fichero.

   c) Vaciar la información que contiene el fichero.

   d) Permite el acceso al fichero a un solo usuario.

Complete las frases siguientes:

12. Los dispositivos de memoria auxiliar se utilizan para .... la información.

13. El formato de la información almacenada en los dispositivos de memoria auxiliar es .... por el ordenador.

14. Un .... es un conjunto de datos almacenados en un dispositivo de memoria auxiliar y dispuestos con una organización determinada.

15. La estructura de la información es de tipo ....

16. Una sucesión de datos o caracteres que contienen una información determinada se denomina ....

17. Un conjunto de campos que forman una unidad lógica se denomina ....

18. Una colección de información relacionada lógicamente se denomina ....

19. El tamaño de un registro se mide en ....

20. Un archivo de ficheros se denomina ....

21. Cuando un fichero tiene sus registros dispuestos en forma sucesiva se llama ....

22. Los ficheros .... se llaman también relativos o de acceso al azar.

23. Los ficheros a los que se accede mediante una clave, se denominan ....

24. Para poder acceder a la información que contiene un fichero desde un programa se debe realizar una operación de ....

25. Cuando se accede a un dispositivo de almacenamiento se lee de una vez todo un ....

26. Para recuperar la memoria intermedia utilizada para trabajar con un fichero, se debe realizar la operación de ....



17.4 INICIACIÓN AL LENGUAJE MÁQUINA

Nivel de los lenguajes

Internamente, todos los ordenadores trabajan siempre con unas instrucciones elementales. El conjunto de instrucciones aceptadas por una máquina constituye el llamado *lenguaje máquina* o también *código máquina*. Si bien es cierto que todos los ordenadores operan bajo el código máquina, sería muy difícil para los usuarios construir programas en este lenguaje. Por esta razón se desarrollaron una serie de lenguajes de programación que facilitaran la comunicación con la máquina. Estos lenguajes, en el fondo no son más que unos programas que traducen unas instrucciones escritas en una sintaxis próxima a nosotros a código máquina. El grado de proximidad de un lenguaje de programación al lenguaje natural del hombre se denomina *nivel*. Así, un lenguaje de bajo nivel está cercano a la máquina. El de más bajo nivel es precisamente el lenguaje máquina. Por el contrario, un lenguaje de alto nivel estará más cerca del lenguaje humano. El BASIC constituye un ejemplo de lenguaje de alto nivel. Una instrucción cualquiera de BASIC, por ejemplo:

```
10 LET B=4+2*A
```

es fácil de entender. En cambio, su traducción a código máquina da lugar a un conjunto grande de pequeñas instrucciones elementales, difíciles de descifrar.

En el capítulo siguiente estudiaremos más detenidamente todo lo que hace referencia a los distintos lenguajes de programación. En este apartado nos centraremos en el lenguaje de la máquina.

Dejando aparte las dificultades de programar en código máquina, la ventaja de este lenguaje es que nos permite obtener la máxima eficiencia y rapidez del ordenador. Este punto es especialmente importante cuando la velocidad de proceso es un factor crítico (recordemos el tema de los gráficos en movimiento). Otra ventaja es que nos permite acceder a recursos especiales de la máquina que de otra manera nos quedaría vedada su utilización.

Para entender bien cómo opera el código máquina es necesario conocer previamente cómo funciona por dentro un ordenador. En algunas

lecciones anteriores ya tratamos un poco este tema. Sin embargo, entonces tuvimos que limitarnos a dar una visión superficial. Ha llegado el momento de describir internamente al ordenador de una forma más profunda.

17.4.1. Descripción del ordenador

Como ya conocemos, todo equipo informático (ordenador más periféricos) tiene el esquema funcional que se describe en la figura 11. Este esquema ya fue diseñado por Von Neumann, matemático húngaro (nacionalizado estadounidense) que fue el padre de la teoría de las computadoras. En esta sección nos concentraremos en el estudio de la unidad central de proceso o CPU por sus siglas en inglés (Central Processing Unit).

Hemos repetido numerosas veces que el ordenador trabaja internamente en el sistema binario. Convendrá pues, recordar los conocimientos adquiridos en la lección segunda sobre este sistema de numeración. Sin embargo, el sistema binario tiene para nosotros un gran inconveniente. Al tener una base de numeración tan pequeña (el número 2), para representar una cantidad necesitamos un número muy elevado de cifras. Por ejemplo, el número 64 decimal requiere nada menos que 6 cifras en binario. Además, a medida que las cantidades crecen, la situación empeora. Para representar el 35000 se requieren 15 cifras en binario. Esto en cuanto a los números. Pero respecto a los textos la situación es peor, si cabe. Recordemos que cada símbolo necesitaba un byte para almacenarlo, o sea 8 bits. Por tanto, la palabra CASA, por ejemplo, necesitará $4 \times 8 = 32$ bits o cifras binarias. Podemos imaginar fácilmente lo difícil que resultaría para nosotros el descifrar la representación binaria de un texto algo mayor como es el caso de un nombre o una dirección.

Es obvio que necesitamos una solución a fin de representar de forma cómoda las largas series de cifras binarias. Esta solución nos la da el sistema hexadecimal.

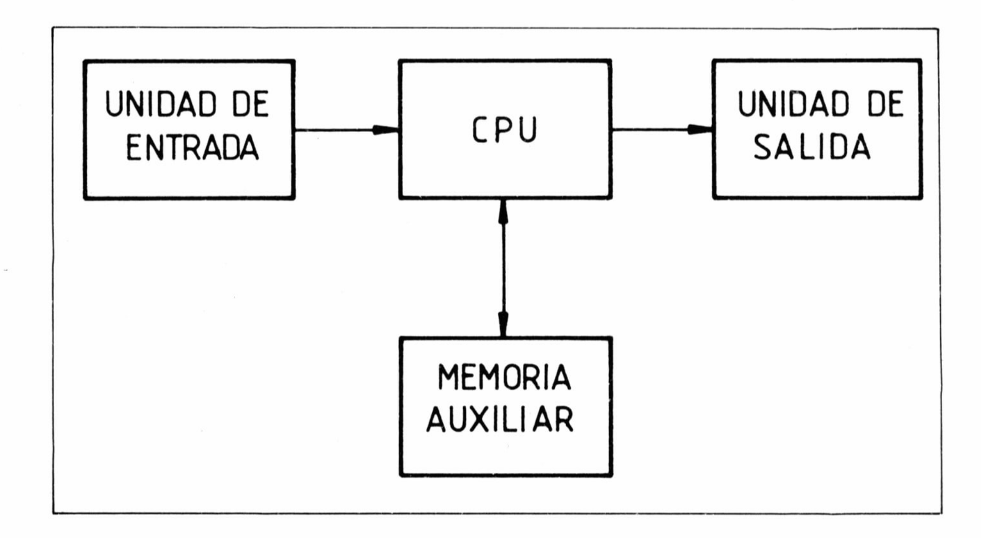


Figura 11. Esquema funcional de un equipo informático.

17.4.2 Sistema hexadecimal

Se necesitaba un sistema de numeración que reuniera estas dos características:

a) Que la representación de una cantidad se hiciera con un número reducido de cifras.

b) Que la conversión a binario (y viceversa) fuera inmediata y sin requerir operaciones.

De la primera característica se deduce que la base de numeración ha de ser grande (al menos mayor de 10). De la segunda se deduce que esta base debe ser un número que equivalga a «2 elevado a algo». Ejemplos de estos números serían 4 (2 elevado a 2), el 8 (2 elevado a 3), el 16 (2 elevado a 4), el 32 (2 elevado a 5), etc.

El hecho de que una posición de memoria esté formada por un byte que se puede dividir en dos grupos de 4 bits, hizo especialmente atractiva la elección del número 16 como base. El sistema de numeración en base 16 recibe el nombre de hexadecimal, que proviene de «hexa» (seis) y «decimal» (diez).

Sistema hexadecimal

Asociado a todo sistema de numeración va el repertorio de dígitos utilizables. El sistema binario tiene 2 (el 0 y el 1), el sistema decimal tiene 10 (del 0 al 9) y el sistema hexadecimal necesita 16. Aquí surge una dificultad. No disponemos de dígitos para completar el repertorio de 16. Una solución podría ser inventarnos los 6 símbolos que faltan a partir del 9 hasta el 15. Esta solución hubiera tenido problemas con los teclados ya existentes. Por tanto, se decidió emplear una solución más sencilla que consiste en tomar prestadas las primeras letras del alfabeto para representar los dígitos que nos faltan. Así, la letra A representa el 10, la B el 11, etc.

La gran ventaja de este sistema de numeración es que cada cifra representa un conjunto de 4 bits. En la tabla de la figura 12 se muestra la equivalencia entre cifras hexadecimales y bits. Observemos que, tal como pretendíamos, hay una reducción en el número de cifras empleadas. El número 12 decimal, en hexadecimal se representa con una sola cifra (la C). Por otra parte, la conversión a binario se hace por simple sustitución de un grupo de 4 bits por la correspondiente cifra hexadecimal. En la figura 13 se muestran algunos ejemplos. El número 93, en binario es 01011101. Los cuatro primeros bits equivalen a un 5 hexadecimal y los cuatro siguientes a una D. Por tanto, el número hexadecimal es el 5D. Podemos comprobar que es cierto haciendo la conversión a decimal:

$$\begin{aligned} 5 \times 16 &= 5 \times 16 = 80 \\ D \times 16 \quad D &= \ \ 1 = \underline{13} \\ & \qquad\quad\ \ 93 \end{aligned}$$

Recordemos, pues, que el contenido de un byte se puede representar mediante dos cifras hexadecimales. Estas cifras variarán desde 00 (los 8 bits a cero) hasta FF (los 8 bits a 1).

Figura 12. Tabla de conversión de hexadecimal a binario.

DECIMAL	HEXADEC.	BINARIO
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Aunque el sistema hexadecimal emplee algunas letras como cifras, no hay que olvidar que se trata de un sistema de numeración como los demás y que por tanto sigue las mismas normas. En consecuencia, después de F (un 15) vendrá el número hexadecimal 10 (equivalente al 16 decimal). Esto es lógico, puesto que al llegar a F hemos agotado el repertorio, al igual que al llegar a 9 en decimal, o a 1 en binario. Entonces hay que poner un cero en la primera columna y empezar la numeración en la siguiente. Detrás de 10 (hexadecimal) siguen 11, 12, 13,... Puesto que si no aparecen letras en el número es imposible distinguir un número decimal de uno hexadecimal. En lo que sigue de texto, a los números hexadecimales que se presten a confusión les añadiremos detrás una H entre paréntesis. Por ejemplo 10 (H). Al llegar a 19 (H) no pasa nada especial

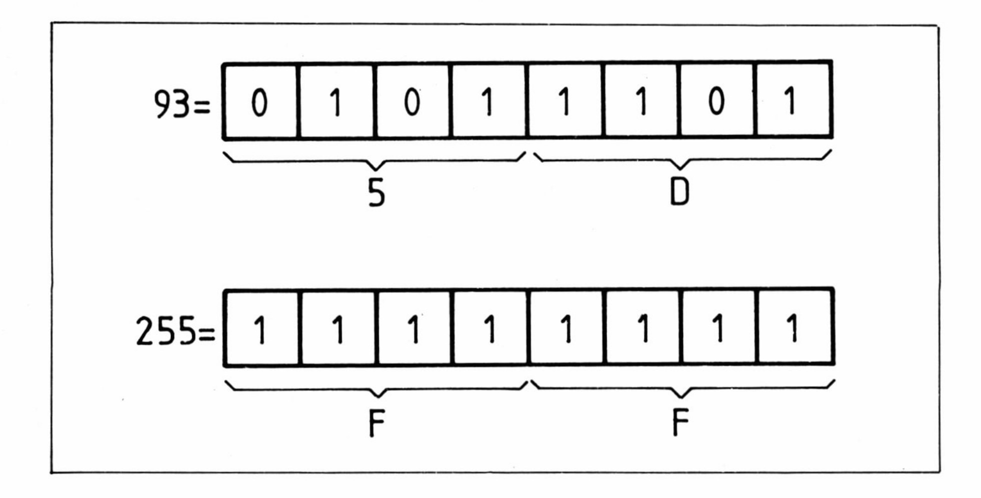


Figura 13. Correspondencia entre el contenido de un byte y su equivalente hexadecimal.

(no hemos agotado aún el repertorio) y sigue por 1A, 1B... hasta alcanzar 1F. En este momento se pasa a 20 (H) y se sigue por 21 (H), etc.

Conversión de hexadecimal a decimal

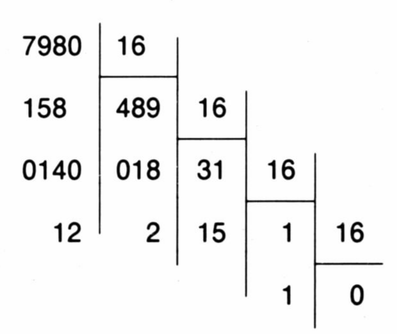
La principal utilización del sistema hexadecimal es la representación del contenido de un byte. No obstante, a menudo se emplea este sistema para indicar valores mayores, como por ejemplo el tamaño de la memoria de la CPU. Por consiguiente, si nos dicen que las posiciones de memoria van desde 0000 (H) hasta 7FFF, necesitaremos saber convertir los valores a sistema decimal. Anteriormente ya hemos realizado una pequeña conversión de este tipo (de 5D a 93). En realidad, la conversión se efectúa de la misma manera que para el sistema binario. Cada cifra se multiplica por la base (en este caso 16) elevada a un exponente que coincide con su posición relativa (empezando por cero).

Tomemos por ejemplo el número 7FFF. En decimal será

$$
\begin{aligned}
7 \times 16^3 &= 7 \times 4.096 = 7 \times 4.096 = 28.672\\
F \times 16^2 &= F \times 256 = 15 \times 256 = 3.840\\
F \times 16^1 &= F \times 16 = 15 \times 16 = 240\\
F \times 16^0 &= F \times 1 = 15 \times 1 = \underline{15}\\
& \qquad\qquad\qquad\qquad\qquad\quad 32.767
\end{aligned}
$$

Conversión de decimal a hexadecimal

El resultado es, pues, 32.767. La conversión de decimal a hexadecimal se realiza por el procedimiento de divisiones sucesivas por la base (16 en este caso). El resultado se formará mediante los restos de las divisiones en orden inverso. Por ejemplo, para convertir 7.980 haremos las siguientes operaciones:



Los restos obtenidos de derecha a izquierda son 1, 15, 2 y 12. Escribiéndolos en cifras hexadecimales, el resultado es 1F2C.

17.4.3 Organización interna de la CPU

La CPU es el núcleo donde se procesan los datos. Consta de una memoria central, de unos circuitos operativos, de unos canales de comunicación y de un reloj que sincroniza todo el proceso. En la figura 14 se

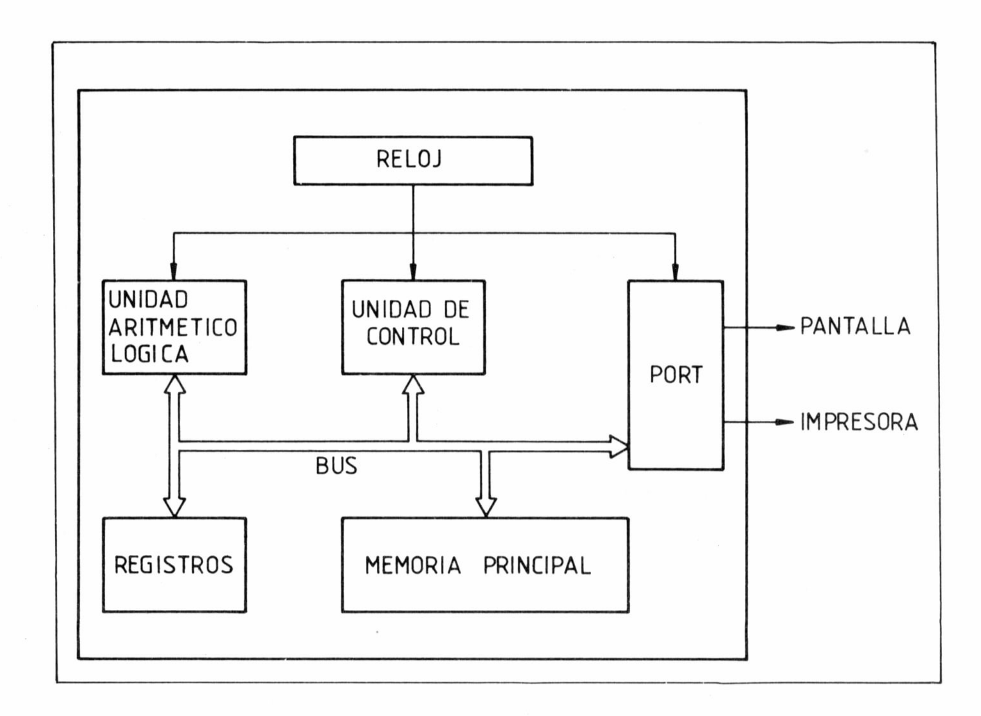


Figura 14. Esquema de los componentes de la CPU.

muestra un esquema de organización de la CPU. En las secciones siguientes se verá una descripción más detallada de cada componente.

17.4.3.1 Unidad de control

Es la unidad encargada del gobierno de las diversas unidades. Ella es la que interpreta y efectúa las instrucciones de un programa. Constituye, por tanto, el núcleo fundamental de la CPU. La unidad de control tiene dos estados diferenciados:

a) Un estado en el cual lee la instrucción contenida en la memoria.

b) Un estado en el cual ejecuta la instrucción.

Cuando se describan el resto de componentes de la CPU daremos más detalles sobre cómo se realiza la búsqueda y ejecución de las instrucciones. Estos dos estados se van repitiendo indefinidamente, sincronizados por un reloj hasta encontrar una instrucción de fin de proceso.

17.4.3.2 Reloj

En realidad, mas que de un reloj, se trata de un generador de impulsos que sincroniza el funcionamiento de los componentes de la CPU. Para los aficionados a la electrónica, diremos que se construyen a base de osciladores de cristal de cuarzo.

Lógicamente cuanto más veloz sea el reloj, más rápidamente se efectuarán las instrucciónes. Es, por tanto, uno de los factores a tener en cuenta para determinar la calidad de una CPU. Las velocidades o frecuencias de oscilación van desde 1 MHz (un impulso cada millonésima de

segundo) hasta 18 MHz (un impulso cada 55 mil-millonésimas de segundo). Estas enormes velocidades explican el hecho de que los ordenadores sean tan rápidos aunque trabajen con unidades de información tan pequeñas como el bit o el byte.

17.4.3.3 Unidad aritmético-lógica

También se denomina ALU por sus siglas en inglés (Arithmetic and Logic Unit). Esta unidad, bajo el gobierno de la unidad de control, realiza las operaciones aritméticas (suma, resta, multiplicación y división), las operaciones relacionales (comparaciones de igualdad y desigualdad) y las operaciones lógicas (operadores AND, OR, etc.) con los datos. Estos datos están contenidos siempre en los registros.

17.4.3.4 Registros

Registro como grabación

Antes de seguir con la explicación hay que aclarar algunas cuestiones sobre la nomenclatura. En español se utiliza la palabra registro para expresar dos conceptos distintos. El primer significado de registro equivale al de la palabra inglesa «record» y se refiere a «aquello que ha sido grabado». Este es el significado que hemos explicado al hablar de ficheros. La segunda acepción de registro equivale al de la palabra inglesa «register» y significa «posición de memoria especializada» y no tiene ninguna relación con el registro de un fichero.

Registro como posición de memoria especializada

Al ser ambos significados tan absolutamente distintos, es fácil deducir por el contexto a qué clase de registro nos referimos. Precisamente, por esta razón, conviene no olvidar la diferencia puesto que en la inmensa mayoría de textos sobre informática no se cita explícitamente la clase de registro del que se está hablando. Esto es especialmente cierto si son traducciones del inglés, ya que en ese idioma no existe la posibilidad de confusión por tener dos palabras distintas.

Los registros de la CPU son posiciones de memoria especializadas. Las unidades de control y aritmético-lógica operan siempre sobre los registros. Para realizar operaciones con los datos que están en la memoria central, hay que transferirlos primero a los registros. Entonces se opera con ellos y se devuelven a la memoria central.

El tamaño de los registros varía desde 8 bits en los ordenadores pequeños hasta los 64 bits de los grandes ordenadores. Este tamaño (llamado a veces tamaño de palabra) es otro de los factores que definen la calidad de una CPU. Todas las operaciones elementales manejan siempre conjuntios de bits del tamaño de un registro. Es evidente que para realizar operaciones con números de muchas cifras (que se almacenarán en varios bytes) los ordenadores con registros pequeños requerirán efectuar operaciones parciales para llegar al resultado final.

Es corriente también que, para algunas operaciones, se usen dos registros unidos a fin de aumentar la eficiencia. Cuando en informática se habla de un ordenador de 8 bits, significa que el tamaño del registro es de 1 byte. Si se indica que un ordenador es de 16/32 bits, significa que trabaja con registros de 2 bytes (16 bits) y que, a veces, utiliza dos registros unidos (4 bytes o 32 bits).

Los registros indispensables para que funcione la CPU son:

a) Contador de programa: Indica a la unidad de control dónde está la siguiente instrucción que debe ejecutarse.

b) Registro de instrucción: La unidad de control deposita en este registro la instrucción que se va a ejecutar y la mantiene allí hasta completar la ejecución.

c) Acumulador: Almacena los datos procedentes de la memoria o procesados por la ALU. Todas las CPUs tienen varios registros acumuladores a fin de que la unidad de control pueda almacenar resultados parciales sin tener que recurrir a la memoria principal.

17.4.3.5 Memoria principal

Es la memoria directamente accesible por la unidad de control. En ella se almacena dos tipos de información. Por una parte, el conjunto de instrucciones que constituyen el programa y, por otra, los datos sobre los cuales operará dicho programa.

Dirección de memoria

En la actualidad, en casi todos los ordenadores independientemente de su tamaño, la memoria principal está formada por varios miles (o incluso millones) de bytes. Cada uno de estos bytes se identifica por su número de orden. Este número, en términos informáticos se denomina *dirección de memoria.* Si decimos, por ejemplo, que almacenamos la palabra CASA en la dirección 4.816, significa que la letra C se almacena en el byte que tiene este número de orden y que las restantes letras se depositan en los bytes siguientes como se ve en la figura 15.

En esta figura, las direcciones de memoria se han representadio en sentido creciente hacia arriba. En informática es frecuente utilizar también esta representación de la memoria.

Memoria volátil y memoria no-volátil

Existen una serie de características que nos permiten clasificar a los diversos tipos de memorias que existen. Una memoria es *volátil* si necesita suministro de energía (corriente eléctrica) para mantener la información. En caso contrario se denomina *no-volátil.*

Memorias dinámicas y estáticas

Las memorias *dinámicas* son aquéllas en las que la información almacenada se degenera con el tiempo y es preciso un refresco periódico para que no se pierda la información. En caso contrario se denominan *estáticas.*

Según su construcción, las memorias pueden ser de ferrita o de semiconductores.

a) Núcleos de ferrita. Son memorias de tipo no volátil. Cada bit se almacena en un pequeño aro de ferrita (polvo de óxidos metálicos compactados) que se magnetiza en una sentido o en otro. Estos aros se disponen en unas parrillas que constituyen las celdas de memoria (Fig. 16). Fueron las primeras memorias que se utilizaron. Su uso es muy restringido actualmente.

Memorias RAM

b) Semiconductores. Existen dos tipos fundamentales de memorias de semiconductores. Como ya se explicó en una lección anterior, estos dos tipos se denominan RAM y ROM. Las memorias de tipo RAM son de uso general y se utilizan para lectura y escritura tanto de programas como

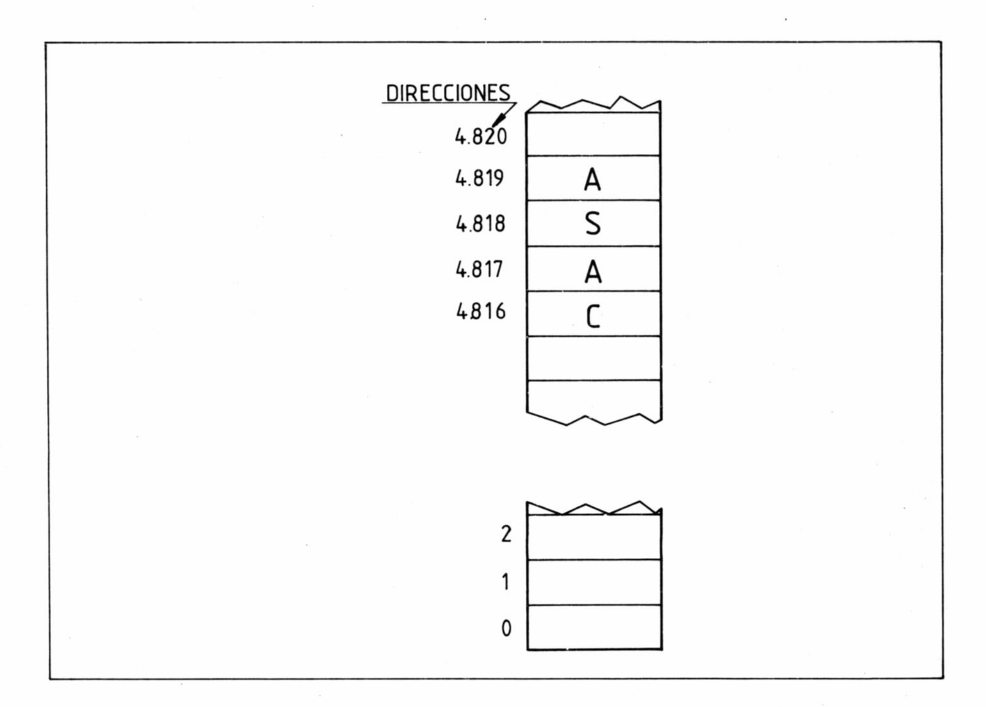


Figura 15. Forma en que se almacena un texto en al memoria central.

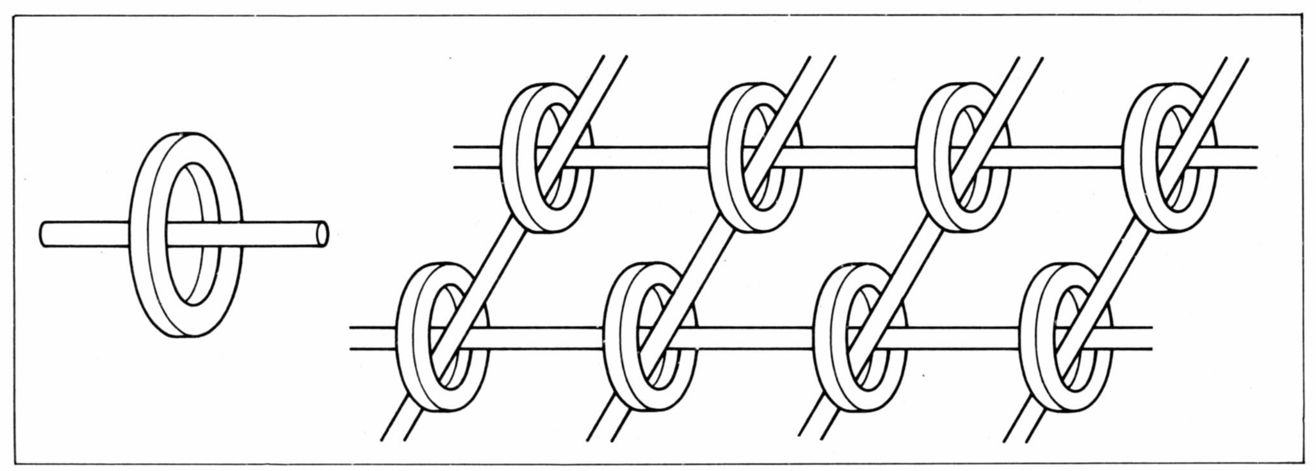


Figura 16. Núcleo de ferrita y celdas de memoria.

de datos. Son de tipo volátil y pueden ser estáticas o dinámicas. No obstante, algunos ordenadores portátiles incorporan memorias RAM no volátiles.

Memorias ROM

Las memorias ROM son no volátiles. Únicamente se pueden utilizar para contener programas o datos fijos. Dentro de este grupo existen las memorias PROM y EPROM. Son memorias cuyo contenido se puede borrar por medios especiales, generalmente mediante la utilización de rayos ultravioleta, y programables de nuevo. PROM significa Programmable Read Only Memory (memoria programable de sólo lectura) y EPROM significa Erasable Programmable Read Only Memory (memoria borrable y programable de sólo lectura).

Las memorias de semiconductores se construyen con el mismo formato que los demás circuitos integrados como se ve en la figura 17.

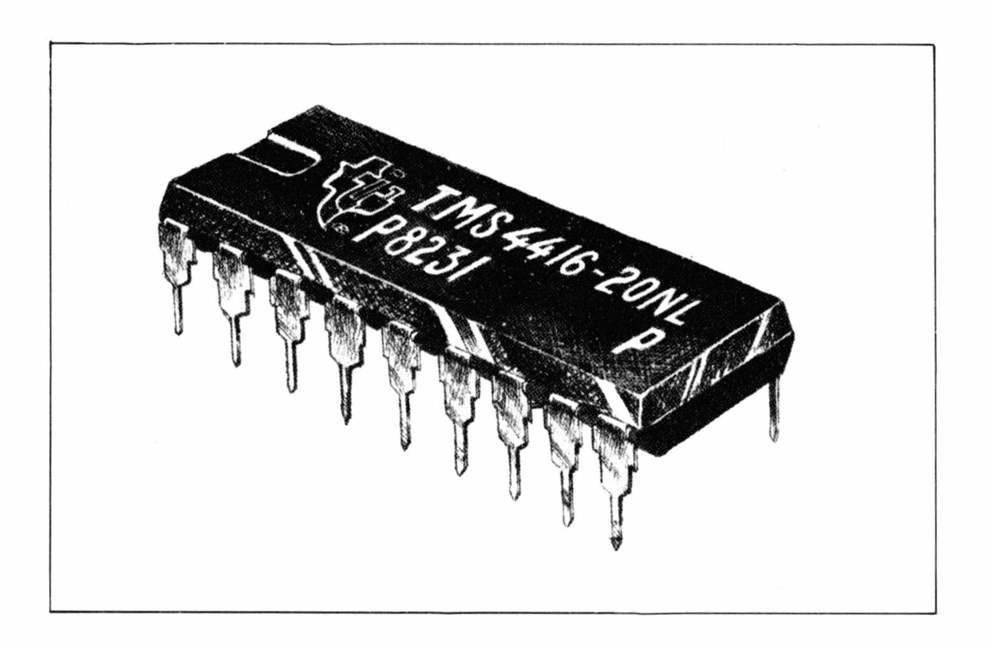


Figura 17. Circuito integrado que contiene una memoria RAM dinámica

17.4.3.6 Canales de comunicación

BUS

La transmisión de datos y órdenes entre los diversos componentes de la CPU se realiza mediante una serie de circuitos o canales de comunicación. También existe un canal de comunicación con el exterior para recibir o transmitir datos a los periféricos. Un canal de comunicación se denomina también *bus*. Un bus transmite al menos un byte completo cada vez. En ordenadores grandes se utilizan canales de 16 y de 32 bits. No siempre se corresponde el tamaño del bus con el del registro sino que, a menudo, es inferior aunque lo ideal sería que coincidieran.

Tipos de bus

Existen tres tipos de bus: *de datos*, *de direcciones* y *de control*. La transferencia de información entre los diversos componentes se realiza a través del bus de datos. El bus de direcciones es el que señala dónde se encuentra el dato (dirección de memoria) que hay que transferir. Finalmente, mediante el bus de control, la unidad de control envía órdenes a las demás unidades de la CPU.

El canal que comunida con el exterior desemboca en los llamados *ports* o *portales* o *puertos*. Un port es el circuito que actúa de receptor (o de emisor) de la información con el exterior. No confundir con una «interface» o interfaz que es un circuito que actúa de adaptador entre CPU y periférico, cuando éstos no son directamente conectables.

17.4.3.7 Concepto de microprocesador

Un microprocesador es un circuito fabricado en una sola pieza que contiene en su interior la unidad de control, la unidad aritmético-lógica (ALU) y los registros. Un microordenador es aquel ordenador cuya CPU contiene un microprocesador (más la memoria, el reloj y los canales de comunicación). En ordenadorès grandes, los diversos componentes están fabricados en circuitos separados, si bien es cierto que la tecnología

actual ha introducido numerosísimas variantes en este esquema tan simple. De hecho, el empleo de microprocesadores es tan amplio que, actualmente, los términos CPU y microprocesador se consideran sinónimos.

Uno de los microprocesadores más usados en microordenadores es el popular Z-80. Se trata de un microprocesador de 8 bits con frecuencia de reloj desde 1 MHz hasta 8 MHz.

17.4.4 Código máquina

Repertorio de instrucciones

El conjunto de instrucciones distintas que la CPU sabe ejecutar se denomina *repertorio de instrucciones.* Los repertorios más corrientes varían entre 150 y 500 instrucciones, las cuales se agrupan en cuatro tipos fundamentales:

a) Instrucciones de Entrada/Salida: Gobiernan los intercambios de datos entre la memoria interna y las unidades periféricas.

b) Instrucciones de Cálculo: Son las que ordenan la ejecución por parte de la ALU de las operaciones aritméticas y lógicas.

c) Instrucciones Logísticas: Se ocupan de las transferencias de los datos entre la memoria principal y los registros y también de una parte a otra de la memoria principal.

d) Instrucciones de control: Son las que modifican el flujo del proceso. Estas instrucciones actúan sobre el registro contador de programa.

Cuanto más completo sea el repertorio de una CPU más potente será el ordenador que la incorpore. A menudo se potencia un grupo determinado de instrucciones según el tipo de utilización de un ordenador. Así, un ordenador de gestión tendrá un número elevado de instrucciones de Entrada/Salida, mientras que un ordenador de uso científico o técnico tendrá un gran número de instrucciones de cálculo.

Sabemos que un ordenador lo único que sabe hacer es seguir al pie de la letra las instrucciones que le demos; no es capaz de ir más lejos pero, sin embargo, presenta la ventaja de seguir las instrucciones de manera fiel y extraordinariamente rápida. Estas instrucciones se dan bajo la forma de un programa que, como ya sabemos, se reduce siempre a un conjunto de instrucciones adecuadamente ordenadas.

El programa se almacena en la memoria. Cada instrucción del lenguaje máquina ocupa una o varias posiciones de memoria y está codificada con un conjunto de bits. Las instrucciones siguen un formato donde se asignan zonas para identificar el tipo de instrucción y zonas para indicar los detalles sobre los operandos que maneja la instrucción (pueden ser ninguno, uno o varios). Estas zonas reciben el nombre de *zona de código de operación y zona de operandos* (Fig. 18).

Tomaremos como ejemplo de utilización, el repertorio de instrucciones del Z-80 por ser el más empleado en ordenadores pequeños.

Aunque no tenemos interés en convertirnos en especialistas en código máquina, no estará de más conocerlo un poco de forma superficial. De esta manera comprenderemos mejor cómo funciona la máquina por

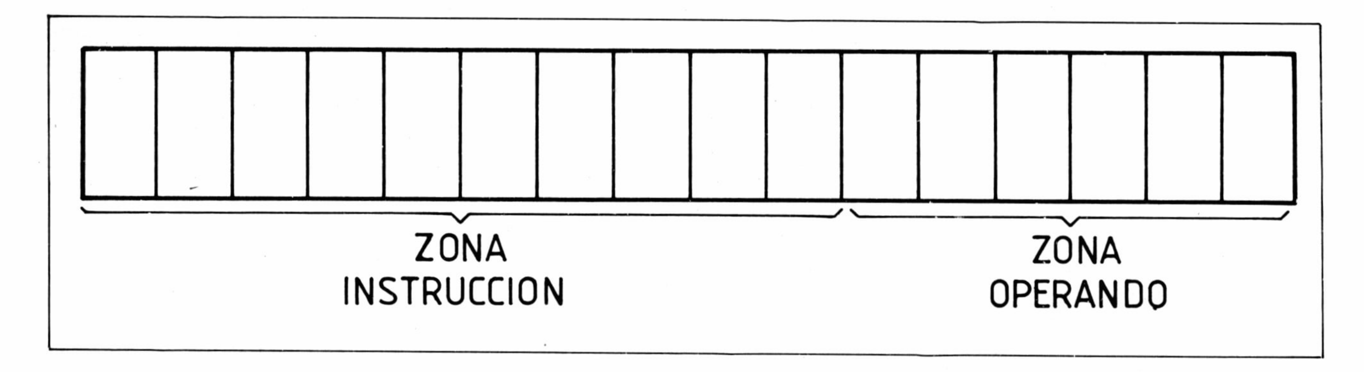


Figura 18. Zona en que se divide una instrucción en código máquina.

Dirección memoria	Código máquina	Descripción
0000	3A6400	Cargar el registro A con el contenido de la dirección 100 (64 Hex.). Es una instrucción de tres bytes.
0003	47	Cargar el registro B con el contenido de A.
0004	3A6500	Cargar el registro A con el contenido de 101.
0007	80	Sumar los registros A y B.
0008	326600	Almacenar el resultado en la dirección 102.
0064	1F	Sumando 1.
0065	2A	Sumando 2.
0066	0	Posición donde se almacena el resultado.

Figura 19. Segmento de programa que suma los contenidos de dos posiciones de memoria.

dentro y, llegado el caso, podremos entender o incluso construir un trozo de programa especial escrito en dicho código. En la figura 19 se lista un segmento de programa que suma el contenido de la posición de memoria 100 con la 101 y lo almacena en la 102.

Analicemos con más detenimiento el programa anterior. En primer lugar, las direcciones de las posiciones de memoria se expresan mediante dos bytes. Al utilizar 16 bites tenemos la posibilidad de numerar hasta 65.536 posiciones distintas (de 0 a 65.535). Este número se calcula elevando 2 a 16 (es decir, multiplicando 2 por 2 dieciséis veces). Estas 65.536 posiciones equivalen exactamente a 64 Kby. Precisamente ésta

es la razón por la cual los ordenadores pequeños suelen tener limitada la memoria central a 64 Kby.

Las direcciones están escritas en hexadecimal para poderlas relacionar fácilmente con las instrucciones (segunda columna de la tabla). En la figura 20 se puede ver la equivalencia entre las direcciones expresadas en decimal y en hexadecimal.

Ya hemos mencionado anteriormente que una instrucción de código máquina ocupa una o varias posiciones de memoria. Precisamente, la primera instrucción del programa ocupa tres posiciones. El primer byte indica que se va a realizar (código de operación). Para el caso concreto del microprocesador Z80, el código 3A significa traer un byte de la memoria sobre el primer registro (llamado registro A). Los bytes siguientes, cuyos valores respectivos son 64 y 00 (hexadecimal) contienen la dirección de memoria. Hay que advertir que en el Z80 estos están siempre en orden inverso. Por esta razón tenemos 6400 y no 0064 como parecería logico. En resumen, estos tres bytes indican a la unidad de control que debe traer el contenido de la posición de memoria 0064 (hex.) o 100 (decimal) al registro A. Recordemos que todas las operaciones se realizan siempre sobre los registros. El valor que se ha trasladado es el 1F (hex) o 31 en decimal.

A continuación, el código 47 es la orden para realizar una copia del contenido del primer registro (el A) sobre el segudo (el B). En la figura 20 tenemos la equivalencia en binario de todos estos códigos. Por tanto, la parte derecha de la figura refleja con toda exactitud el contenido de la memoria del ordenador. Podríamos preguntarnos el por qué no hemos

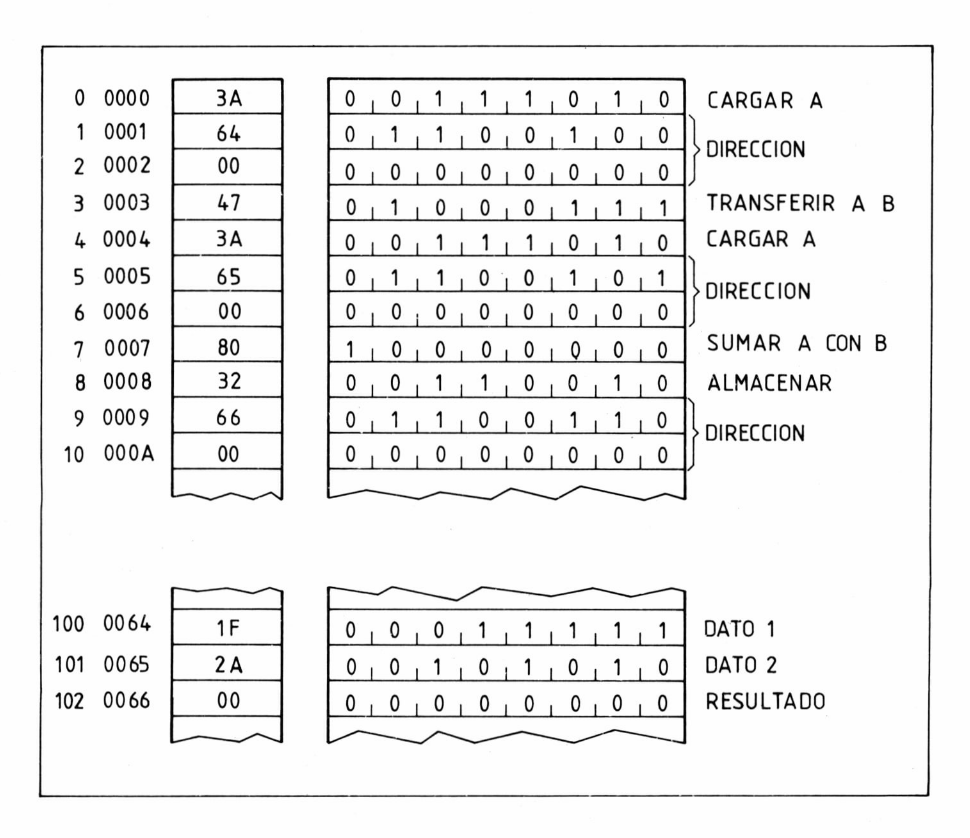


Figura 20. Distribución exacta en la memoria del programa en código máquina.

trasladado la posición de memoria 100 directamente sobre el registro B. La razón es que el repertorio del microprocesador Z80 es limitado y no tiene esta instrucción. Esta carencia obliga a efectuar un pequeño rodeo utilizando el registro A. Esta instrucción (el código 47) ocupa una sola posición de memoria, a diferencia de la anterior que ocupaba 3.

La siguiente instrucción es de nuevo un 3A que significa, como ya conocemos, traer un dato sobre el registro A. En este caso se traslada el segundo sumando que se encuentra en la dirección 0065 (hex.) o 101 en decimal. Su valor es 2A (hex.) o 42 decimal. Fijémonos que los tres bytes que componen esta instrucción tienen la misma estructura que los tres primeros que hemos visto.

Ya tenemos, pues, los registros A y B que contienen los dos sumandos. Ahora se efectúa la operación de suma que viene indicada por el código 80. El resultado queda almacenado en A. De hecho, el programa podría terminar aquí. Pero si estas instrucciones formasen parte de un programa mayor, entonces las siguientes instrucciones necesitarían utilizar este registro ocasionando, por tanto, la pérdida del valor obtenido. Para evitarlo, hay que sacarlo del registro y almacenarlo en la memoria principal. Destinamos la posición 102 (decimal) para ello. La instrucción 32 indica que se transfiere el contenido de A a la posición de memoria indicada en los dos bytes siguientes. De nuevo tenemos una instrucción de tres bytes.

Como se puede apreciar, una operación tan simple como la suma de dos bytes requiere varios pasos. Es fácil imaginar la dificultad que tendrá la elaboración de un programa que efectúe operaciones más complejas.

Puesto que es muy difícil recordar el significado de los códigos, los programadores utilizan un lenguaje especial, llamado ASSEMBLER, para elaborar un programa en código máquina. Este lenguaje asocia unos símbolos a los códigos numéricos para que sea más fácil de recordar. Por ejemplo, el programa anterior, escrito en ASEMBLER sería:

```
LD   A,(100)
LD   B,A
LD   A,(101)
ADD  A,B
LD   (102),A
```

En donde LD es la abreviatura de LOAD (cargar en inglés) y ADD significa adicionar o sumar. Es muy fácil relacionar estas instrucciones escritas así con las explicaciones precedentes.

17.4.5 Función PEEK

Función PEEK

En la gran mayoría de microordenadores, cuando los ponemos en marcha entramos directamente en el lenguaje BASIC. Si bien es cierto que desde este lenguaje podemos acceder a la memoria de la máquina a través de las variables, no es menos cierto que nunca sabemos en que

posición concreta se encuentran los datos almacenados. Sin embargo, existe en BASIC unas funciones que nos permiten consultar y modificar una posición concreta de la memoria. La primera de ellas es la función PEEK cuya sintaxis es:

```
PEEK(n)
```

en donde *n* es la posición de memoria a consultar. En otras palabras, es la dirección del byte. La palabra inglesa PEEK significa «recoger». El resultado de esta operación es el contenido de byte «n». En consecuencia, el resultado estará comprendido entre 0 y 255. Una forma típica de utilización sería:

```
PRINT PEEK(8000)
```

Puesto que ahora consultamos la memoria directamente (en este caso la posición 8000) sin intervención del BASIC, el contenido dependerá de cada modelo de ordenador. Además, si la posición consultada pertenece a la zona donde trabaja nuestro programa, el contenido variará según el momento en que realicemos la consulta.

De lo expuesto anteriormente se deduce que la utilización de este tipo de instrucciones, si bien es interesante para acceder a algunos recursos de la máquina, hará que los programas sean de difícil adaptación a otros modelos de ordenador.

El siguiente programa nos dará el contenido de las 10 primeras posiciones de memoria

```
10 FOR I=1 TO 10
20    PRINT I,PEEK(I)
30    NEXT I
```

17.4.6 Instrucción POKE

La operación complementaria de PEEK (recoger) será la de almacenar un dato en una posición de memoria. Esta instrucción recibe el nombre de POKE que, significa «hurgar», aunque en este caso tenga más bien el sentido de colocar.

La sintaxis general es:

```
POKE n,m
```

Instrucción POKE

en donde *n* es la posición de la memoria y *m* es el dato a almacenar. En el fondo, guarda un cierto parecido con la instrucción LET ya que ambas almacenan un dato en la memoria. Sin embargo, la instrucción POKE sólo

puede almacenar un único byte. A cambio, puede almacenarlo en cualquier posición de la memoria que deseemos.

No olvidemos, por otra parte, que POKE es una instrucción completa a diferencia de PEEK que es una función. Por tanto, PEEK *debe* emplearse dentro de una instrucción de BASIC como, por ejemplo, PRINT, LET, etc. Por el contrario, POKE *no puede* emplearse dentro de otra instrucción, ya que ella misma lo es.

Así como la función PEEK era totalmente inofensiva y podíamos utilizarla sin peligro, la instrucción POKE es muy delicada. Si se altera el contenido de ciertas posiciones de memoria vitales, el ordenador quedará bloqueado y nos veremos obligados a ponerlo en marcha de nuevo (afortunadamente, los datos nunca son permanentes y basta con desconectar el ordenador y volverlo a conectar para que funcione de nuevo).

Importante: Antes de utilizar la instrucción POKE debemos asegurarnos de que no alteramos una posición de memoria reservada para el uso interno del ordenador.

Al permitir el acceso directo a la memoria, estas dos instrucciones son utilizadas para realizar algunas operaciones especiales, fundamentalmente de tipo gráfico. Sin embargo, debido a que su empleo concreto depende de cada modelo de ordenador, reservaremos su estudio más detallado para las prácticas con el microordenador. Concretamente, las prácticas del capítulo siguiente.

17.4.7 Función USR

Mediante la instrucción POKE colocaremos un programa en código máquina en la memoria. Falta ahora un paso importante: ¿Cómo indicar al ordenador que debe ejecutar dicho programa?

En los programas escritos en BASIC la solución consiste en utilizar el comando RUN. Sin embargo, como es lógico, este comando no sirve para los programas en código máquina. La forma más corriente en BASIC, para pasar control a un programa de este tipo, es utilizando la función USR. La sintaxis general es:

```
USR(n)
```

Función USR

en donde *n* es la posición de memoria donde empieza nuestro programa. Hay que advertir que en algunas versiones de BASIC esta sintaxis es algo distinta. La palabra USR proviene de la abreviatura de «user subroutine» que significa subrutina del usuario. La utilización de la palabra subrutina en lugar de programa se justifica porque, en la práctica, estos programas se escriben en forma de subrutina. De esta forma, cuando han terminado, devuelven control al programa en BASIC que actúa así de programa principal. Por el contrario, si el programa en código máquina no es una subrutina, cuando termine la ejecución la máquina se bloquea-

rá, puesto que la CPU dará por terminada cualquier tarea incluyendo al propio BASIC. No olvidemos que cuando estamos trabajando en código máquina tenemos un control absoluto sobre el ordenador.

Una forma típica de utilizar la función USR es con la instrucción PRINT. Por ejemplo:

```
PRINT USR(16300)
```

Con esta instrucción pasaríamos control a la subrutina almacenada en la posición 16300. Como toda función, USR nos devuelve un valor. Depende de cada modelo de ordenador el significado de este resultado. A menudo corresponde al contenido de un registro.

17.5 Sistemas operativos

Un ordenador recién fabricado está formado por un conjunto de circuitos electrónicos. Estos circuitos tienen la propiedad de que son programables, pero inicialmente no «saben» hacer nada, es decir, son sólo una aglomeración de piezas (lo que se denomina hardware en lenguaje informático). Para hacer un uso eficiente de la máquina es necesario programar dichos circuitos para que sean capaces de realizar las tareas más básicas.

Estamos acostumbrados a que pulsando una tecla aparezca en pantalla la letra escrita, pero, ¿nos hemos preguntado alguna vez cómo sabe el ordenador que la letra A, por ejemplo, tiene esta forma y no otra? La respuesta está en que se han programado dichos circuitos (en código máquina) para que sepan dibujar las letras de nuestro alfabeto. En otros países, como por ejemplo Japón, estos programas incluyen otros tipos de alfabetos distintos al nuestro.

Primer nivel del Sistema operativo

Análogamente, deberán existir otros programas o subrutinas para controlar todos los periféricos, como la impresora, la grabadora de cassette, etc. Este conjunto de programas constituyen lo que se llama núcleo de un sistema operativo. Este núcleo recibe, a veces, el nombre inglés de KERNEL. La existencia de un sistema operativo es común a todos los ordenadores, tanto si se trata de un pequeño microordenador personal como si se trata de un gran equipo con miles de terminales aunque, obviamente, las capacidades de uno y otro serán muy distintas.

Segundo nivel al sistema operativo

Si el sistema operativo contuviera solamente el núcleo, las posibilidades de utilización de la máquina quedarían muy reducidas. En realidad sólo podríamos programar en código máquina. Por esta razón, el sistema operativo incorpora un segundo nivel. En este nivel encontramos las siguientes posibilidades:

a) Lenguajes de programación. Dispondremos de uno a más lenguajes (BASIC, FORTRAN, PASCAL,...) para desarrollar nuestros propios programas.

b) Programas de servicio. Los programas de servicio («utility» en inglés) se encargan de arropar al sistema operativo. Su cometido es

realizar procesos que permiten agilizar el resto de tareas del sistema, mejorando así el rendimiento global del equipo. Programs típicos de este tipo son: Copia de seguridad, clasificadores de ficheros, depuradores de errores en los programas y programas de diagnóstico de averías.

c) Programas de aplicación. Son programas destinados a realizar una tarea concreta de interés para el usuario final, por ejemplo, una contabilidad, un cálculo estadístico, etc. Estos programas pueden haber sido desarrollados por nosotros mismos o bien ser adquiridos a una empresa de desarrollo de software. El sistema operativo se encargará de que estos programas funcionen en nuestro ordenador y puedan utilizar los periféricos sin problemas.

Tercer nivel del Sistema operativo

Finalmente, el sistema operativo lleva un último nivel que constituye el procedimiento de comunicación con el usuario. En la figura 21 se muestra un esquema de esta estructura. Este último nivel adquiere más importancia cuanto mayor es el equipo. En ordenadores multiusuario, el sistema operativo lleva un control estricto de cada uno de los usuarios, asignándoles a cada uno una zona de trabajo en la memoria y permitiéndoles el acceso a unos periféricos y programas predeterminados. La comunicación con el usuario envuelve a todo el resto de niveles. Por esta razón, a veces se le conoce por el nombre inglés de «shell» que significa concha.

Un sistema operativo es un conjunto de procesos para la explotación de los recursos informáticos, solicitados por las actividades de los usuarios. En resumen, el sistema operativo crea un ambiente en el cual los usuarios pueden preparar programas y ejecutarlos sin tener que entrar en los detalles del hardware del equipo.

Se puede decir entonces que el sistema operativo actúa de intermediario entre los circuitos electrónicos del ordenador y los programas del usuario. Cuanto más sofisticada y potente sea esta interfase, más fácil será la programación y la utilización de la máquina.

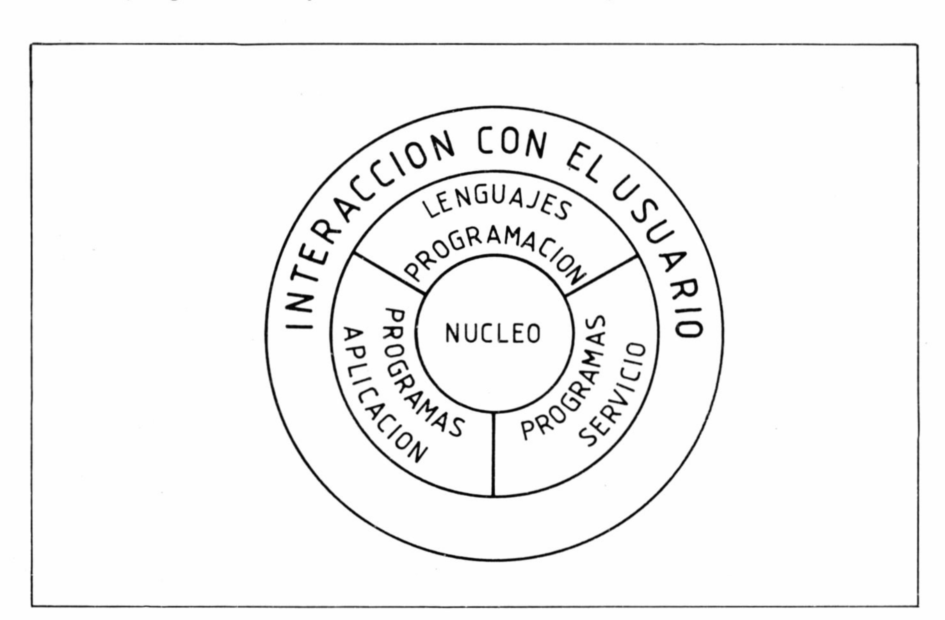


Figura 21. Esquema simplificado de la estructura de un sistema operativo.

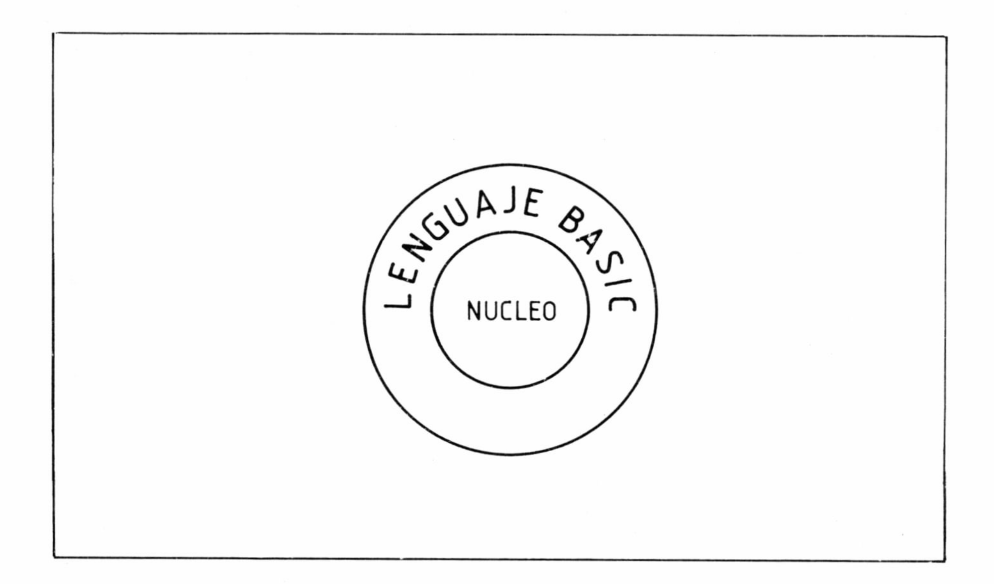


Figura 22. Esquema de la estructura de un sistema operativo de un microordenador.

Anteriormente hemos afirmado que el sistema operativo estaba presente en todas las máquinas. Sin embargo, en el caso de los microordenadores personales queda prácticamente escondido. Como se ve en la figura 22 uno de los lenguajes de programación, el BASIC, invade las demás zonas, constituyendo la zona de comunicación con el usuario y englobando a los programas de servicio y de aplicación. Esta estructura se ha diseñado para que la utilización del ordenador sea más simple aunque, a la vez, las posibilidades queden algo limitadas. En este tipo de sistemas operativos, el BASIC tiene algunas instrucciones especiales que sustituyen a los programas de servicio. Por otra parte, los programas de aplicación se ejecutarán siempre desde el BASIC.

Puesto que en los pequeños microordenadores el sistema operativo queda escondido, podríamos pensar que es un tema de escaso interés. Nada más lejos de la realidad. En primer lugar, porque incluso los más pequeños incorporan un sistema operativo, aunque sea rudimentario y, en segundo lugar, porque si alguna vez usamos una máquina mayor, lo primero que nos encontraremos será la zona de interacción del usuario con el sistema operativo. Probablemente sería algo fustrante que después de dominar un lenguaje de programación tuviéramos la impresión de que sólo somos capaces de manejar nuestro ordenador. En cambio, teniendo algunos conocimientos, aunque sean rudimentarios, sobre los sistemas operativos podremos consultar el manual de la nueva máquina y acceder rápidamente a la zona que conocemos: el lenguaje BASIC.

Finalmente, hay que tener en cuenta que un sistema operativo constituye un modo de funcionamiento y, por consiguiente, un mismo sistema operativo puede estar incorporado a máquinas de distinto modelo y marca. Por otra parte, también existe el caso contrario, es decir, que para un mismo ordenador estén disponibles varios sistemas operativos y entonces se elije aquél que mejor se adapte a las necesidades del usuario. Esta intercambiabilidad entre máquinas y sistemas es cada vez más frecuente, pues se tiende cada vez más hacia una estandarización de sistemas operativos.

Algunos de los sistemas operativos más conocidos para ordenadores pequeños y medianos son:

- Monousuario: CP/M y DOS.
- Multiusuario: MP/M, OASIS y UNIX

UNIX es actualmente el más utilizado en equipos multitarea. Existen muchas variantes del UNIX puro como, por ejemplo, XENIX, IDRIS, etc.

Las prácticas de esta parte del lenguaje máquina las realizará en el capítulo siguiente.

RESUMEN

El conjunto de instrucciones elementales aceptadas por la unidad central del ordenador se denomina código o lenguaje máquina. Este lenguaje es de difícil utilización. Por esta razón se han desarrollado otros lenguajes más cercanos a la expresión humana.

El grado de cercanía al lenguaje humano de un lenguaje de programación se denomina *nivel*. Se denomina lenguaje de alto nivel aquél que está más cerca del sistema de comunicación humano. El lenguaje máquina es el de más bajo nivel (difícil compresión para el hombre), pero con és se obtiene la máxima eficacia del ordenador.

El ordenador trabaja internamente en código binario. Sin embargo, este sistema tiene para nosotros el inconveniente de que se necesitan un gran número de cifras para representar cualquier número, dado que la base de numeración es pequeña: el número 2. Para solucionar este problema, se utiliza el sistema hexadecimal, o de base 16, que tiene la ventaja de representar valores grandes con un número reducido de cifras y de que, además, la conversion a binario es inmediata y no requiere operaciones. El repertorio de dígitos del sistema hexadecimal es: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. Cada cifra en este sistema representa un número de 4 bits. El contenido de un byte se puede representar pues mediante dos cifras hexadecimales. Para convertir un número binario a hexadecimal bastará agruparlo de cuatro en cuatro cifras (de derecha a izquierda) y efectuar la conversión directa.

Por otra parte, para convertir a decimal un número hexadecimal, se multiplicará el valor de cada una de sus cifras por la base (16) elevada al número dado por la posición de esta cifra, contando de derecha a izquierda, empezando por cero. La conversión inversa (decimal a hexadecimal) la realizaremos efectuando las divisiones sucesivas por 16 hasta obtener el cociente cero, y tomando los valores de los restos de cada división empezando por la última efectuada.

El núcleo donde se procesan los datos del ordenador se denomina CPU, o unidad central de procesos, y consta de distintas partes. Dentro de la CPU, la unidad encargada del gobierno de las

demás se denomina unidad de control. Esta unidad va pasando por dos estados (lectura de instrucción, ejecución de la misma), sucesivamente, sincronizada por el reloj o generador de impulsos, que va marcando el ritmo del funcionamiento de todo el ordenador. Así, cuanto más rápido sea el reloj, más rápidamente se ejecutarán las instrucciones. La parte de la CPU donde se efectúan las operaciones con los datos se denomina ALU o unidad aritmética-lógica. Estos datos están contenidos siempre en registros. Los registros son posiciones de memoria especializadas, sobre los que operan la unidad de control y la unidad aritmético-lógica. Antes de trabajar sobre un dato, se traslada desde la memoria a los registros, y una vez realizadas las operaciones se devuelven de nuevo a la memoria central. El tamaño de los registros es variable, desde 8 bits hasta 64 en los grandes ordenadores. Este número, conocido como tamaño de palabra, es también una medida de la potencia de un ordenador. Para que funcione el ordenador son indispensables al menos tres registros: el contador de programa, el registro de instrucción y el acumulador. El programa y los datos del mismo se almacenan en la memoria central, donde permanecen mientras lo indique la unidad de control. La memoria central está formada por un número variable (varios miles) de bytes, identificables cada uno de ellos por su número de orden. Este número se denomina dirección de memoria.

Los elementos físicos que constituyen la memoria pueden ser de distintos tipos. Según estos, tendremos memorias volátiles, cuando necesitan suministro continuo de energía, y no volátiles en caso contrario. Si la información almacenada debe regenerarse con el tiempo, las memorias se denominan dinámicas. En caso contrario se denominan memorias estáticas. Según su construcción las memorias pueden ser de ferrita (arc metálico que se magnetiza en uno u otro sentido, de tipo no valátil), o de semiconductores. Las memorias de semiconductores se dividen fundamentalmente en memorias de tipo RAM (para lectura y escritura tanto de datos como de programas), o de tipo ROM (sólo lectura, para programas o datos fijos). Dentro de éstas existen algunas con características especiales: las memorias de tipo PROM, y las memorias de tipo EPROM.

La comunicación entre los distintos componentes de la CPU, y entre ésta y el exterior, se realiza mediante unos canales de comunicación que se denominan bus. Dentro de la CPU existen tres tipos de bus: de datos, de direcciones y de control. El canal que comunica con el exterior para dar o recibir información se denomina «port».

Un circuito que contiene en una sola pieza la unidad de control, la ALU y los registros se denomina microprocesador. Un ordenador cuya CPU contiene un microprocesador (más la memoria, el reloj y los canales de comunicación) se denomina microordenador.

El conjunto de instrucciones distintas que admite la CPU se denomina repertorio de instrucciones. Dentro de él, se encuentran instrucciones de entrada/salida, de cálculo, logísticas y de control. El programa se almacena en la memoria, cada instrucción ocupa una o varias posiciones de memoria, y sigue un formato especial para determinar el tipo de instrucción y los datos sobre los que ha de ejecutarse. Sin embargo, recordar los códigos escritos directa-

mente en código máquina es difícil, y por esta razón se utiliza un lenguaje especial, llamado ASSEMBLER, que asocia símbolos a los códigos numéricos para que sea más fácil de recordar.

En los microordenadores, sin embargo, sólo se puede trabajar en lenguaje BASIC, pero existen dos instrucciones que nos permiten acceder a una posición de memoria concreta. Estas dos instrucciones son PEEK, para leer un valor contenido en un byte determinado de memoria, y POKE, para situar un valor determinado en una posición de la memoria. Mediante esta instrucción se podrá escribir un programa en código máquina en el ordenador. Para ejecutar este programa se utilizará la función USR, que transfiere el control a la posición de memoria donde se encuentra nuestro programa.

Los microordenadores sólo disponen generalmente de esta posibilidad o del lenguaje BASIC. En ordenadores mayores existen un cierto número de programas de apoyo para manipular los distintos recursos del sistema, así como para obtener el mayor provecho del ordenador. Estos programas constituyen el llamado sistema operativo de la máquina, y aunque para los pequeños microordenadores éste es muy rudimentario, conviene tener conocimiento de su existencia dado que nos lo encontraremos siempre, si se nos presenta la posibilidad de trabajar con máquinas mayores.

EJERCICIOS DE AUTOCOMPROBACIÓN

Encierre en un círculo la letra que corresponda a la alternativa correcta:

27. El código máquina es:

a) El conjunto de instrucciones aceptadas por un ordenador.

b) La forma de hablar de las máquinas entre sí.

c) La forma de entrar los datos a los ordenadores.

d) La forma de salir los resultados de los ordenadores.

28. Els sistema binario es:

a) El conjunto de instrucciones que la máquina entiende.

b) Un sistema de numeración en base 2.

c) El formato de los resultados en la unidad de salida.

d) Un sistema de numeración en base 20.

29. La representación en sistema binario del número decimal 100 tiene:

    a) 7 cifras.

    b) 3 cifras.

    c) 2 cifras.

    d) Más de 10 cifras.

30. El sistema hexadecimal utiliza un repertorio de:

    a) 60 dígitos.

    b) 10 dígitos.

    c) 16 dígitos.

    d) 2 dígitos.

31. Número 100 decimal equivale al número hexadecimal.

    a) AO.

    b) 64.

    c) 273.

    d) 10.

32. Para convertir un número decimal en hexadecimal se realiza el procedimiento de:

    a) Divisiones sucesivas por la base 16.

    b) Productos sucesivos del valor por la base elevada a la posición que ocupa el dígito.

    c) Separar las cifras de dos en dos y realizar la conversión directa.

    d) Divisiones sucesivas por la base 10.

33. El valor hexadecimal del número 32654 es:

    a) 7FFF.

    b) 7F8E.

    c) E8F7.

    d) FF7.

34. El valor binario del número hexadecimal 64 es:

    a) 100.
    b) 0010011.
    c) 1100100.
    d) 001.

35. El valor hexadecimal del número binario 11001001 es:

    a) 9C.
    b) 129.
    c) C9.
    d) 1000.

36. El número BBBB es:

    a) Un número hexadecimal.
    b) No es un número.
    c) Un número binario.
    d) Un número decimal.

37. En el sistema hexadecimal se utiliza la letra A como:

    a) Un dígito cuyo valor es 10.
    b) Un dígito cuyo valor es 11.
    c) No se utiliza.
    d) Un dígito que sólo se utiliza en determinadas operaciones.

Complete las siguientes frases:

38. Para sincronizar el funcionamiento de los componentes de la CPU se utiliza un ......

39. Los .... de la CPU son posiciones de memoria especializadas.

40. El tamaño de palabra es el número de .... que tienen los registros de la CPU.

41. El registro que indica a la unidad de control dónde está la siguiente instrucción que debe ejecutarse se denomina ....

42. El registro que almacena los datos procedentes de la memoria o procesados por la CPU se denomina registro ....

43. La memoria directamente accesible por la unidad de control se denomina memoria ....

44. Cada byte de la memoria se identifica por su número de orden, que se llama ....

45. La memoria que necesita suministro de energía se denomina ....

46. El canal de comunicación del ordenador con el exterior se denomina .....

Capítulo 18

• Síntesis del BASIC e introducción a los lenguajes de programación

ESQUEMA DE CONTENIDO

- Objetivos.
  - La síntesis del BASIC.
    - La sintaxis y la semántica.
    - El modo de ejecución inmediato.
    - Los conceptos en la programación en BASIC.
      - Los objetos del lenguaje.
      - Las expresiones.
      - La asignación.
      - Las instrucciones de entrada y salida.
      - Las instrucciones de control.
      - Los datos estructurados.
      - Las funciones.
  - Introducción a los lenguajes de programación.
    - Lenguajes máquina y ensamblador (assembler).
    - Lenguajes interpretados y compilados.
    - Ventajas e inconvenientes.
  - Generalidades de los lenguajes de programación.
    - Los objetos de los lenguajes.
      - Tipos primitivos.
      - Tipos compuestos o estructurados.
        - Tablas.
        - Registros o estructuras.
        - Cadenas de caracteres.
    - Las expresiones.
    - La asignación.
    - Las instrucciones de entrada y salida.
    - Las instrucciones de control.
  - Algunos lenguajes de programación.
    - FORTRAN.
    - COBOL.
    - PL/I.
    - ALGOL.
    - PASCAL.
    - C.
    - Lenguajes interpretados.
      - APL.
      - LISP.
      - LOGD.
  - Los programas de aplicación.
    - Procesadores de textos.
    - La hoja electrónica.
    - Las bases de datos.
  - Conclusión.

18.0 OBJETIVOS

Este capítulo es el último de la Enciclopedia y es obligado en él mirar hacia atrás para luego mirar hacia adelante.

Hay que mirar hacia atrás para reflexionar sobre el contenido de la Enciclopedia y obtener una visión de conjunto que nos haga emerger las características más notables de lo que hemos estudiado. No para acumular nuevos conocimientos sino para asentar definitivamente dentro de un cuadro coherente los conocimientos ya adquiridos.

Se inicia el capítulo con una síntesis del BASIC que nos describe los aspectos del lenguaje en su totalidad. Es posible que encuentre otros BASIC más amplios del que se ha descrito a lo largo de la Enciclopedia. Es necesario saber encuadrar en el lugar oportuno estas ampliaciones.

Además esta visión global nos permite adquirir la facilidad para analizar otros lenguajes ya existentes o incluso los lenguajes que con seguridad aparecerán en un futuro.

La segunda parte es una mirada hacia adelante para ver qué caminos se adivinan, cuáles van a ser las novedades que nos deparará esta ciencia joven que es la Informática. Se inicia esta segunda parte con un estudio de los modos de funcionamiento de los programas que constituyen lenguajes de programación; se introduce y se analiza la diferencia entre un traductor que interpreta y un traductor que compila.

Probablemente el BASIC es el lenguaje más popular, precisamente porque se basa en una interpretación simultánea con la ejecución. Esto presenta grandes ventajas para los que empiezan. Recuerde el significado de las siglas del BASIC. Lenguaje de interés general para principiantes.

El paso siguiente hacia los lenguajes compilados no es tan sencillo de alcanzar. La nueva modalidad de traducción antes de ejecutar hace que nuestra visión de la máquina deba ser mucho más profunda para averiguar con claridad por qué un programa no funciona. Vamos con los ojos tapados a la hora de ejecutar.

No crea, sin embargo, que al salto deba ser muy grande. Ud. ha aprendido los fundamentos y esto le facilitará enormemente la utilización de lenguajes de programación compilados. Sabe juzgar dónde se encuentran las dificultades y esto es un primer paso muy importante.

Se ataca el problema luego de dar los rasgos fundamentales de un lenguaje. Más que darle un conocimiento exhaustivo, se pretende darle el esquema de clasificación para que pueda analizar cualquier lenguaje de programación.

Finalmente y a título informativo describimos unas características muy breves de otros lenguajes de programación.

El estudio de algunas de las aplicaciones más frecuentes en los ordenadores personales es la parte que cierra este capítulo.





18.1 LA SÍNTESIS DEL BASIC

Ha llegado el momento de realizar una visión en perspectiva de lo que es el lenguaje de programación BASIC.

Esta visión nos permite resaltar los aspectos más importantes que, a lo largo de la Enciclopedia, se han diluido en muchos detalles.

Este repaso facilita una posterior ampliación a otros lenguajes. En realidad hay muchos aspectos comunes a otros lenguajes de programación.

18.1.1 La sintaxis y la semántica

La *sintaxis* es aquella parte del lenguaje que explica *cómo hay que escribir* las cosas.

Si se compara con el lenguaje corriente, la ortografía y la gramática nos enseñan a escribir bien las palabras y a formar correctamente las frases. Del mismo modo, en el lenguaje de programación debemos mantener unas reglas para escribir correctamente; por ejemplo, cuando se escribe la instrucción IF, decimos que a continuación debe escribirse una condición, después la palabra THEN y, finalmente, una secuencia de instrucciones. Esta descripción constituye la sintaxis del lenguaje.

La *semántica*, por otra parte nos indica cuál *es el significado* de lo que escribimos. En el lenguaje natural cuando decimos: la mesa es redonda, se enuncia una de las propiedades del objeto mesa y gracias a nuestro conocimiento del significado de las palabras asociamos a una construcción sintáctica un significado. En otros idiomas la manera de expresar este significado es distinto, se utilizan otras palabras y también otras formas de construcción para expresar la misma idea. Sin embargo, la propiedad de la mesa se mantiene, pues es independiente de la manera de expresarlo.

En el lenguaje de programación esta misma idea se repite, en la instrucción

```
IF a<0 THEN LET a = -a
```

Expresamos que si el objeto llamado *a* es menor que cero entonces hay que cambiarle de signo. La forma de expresarlo es particular del BASIC. Sin embargo, el significado pertenece a una operación obligada por el problema a resolver.

Los errores sintácticos

Es fácil intuir que los aspectos más importantes son precisamente los semánticos; es decir, el significado exacto de lo que queremos hacer. La forma de expresarlos es necesaria, pues en toda comunicación hay que expresar los significados de alguna manera. Pero, realmente la importancia de la forma es secundaria.

En BASIC los errores nos indican cuándo las cosas no se hacen bien. Es difícil decir si este error proviene de un aspecto sintáctico o de un aspectos semántico. Por ejemplo, muchas veces en BASIC aparece el error

Syntax error

que significa claramente que en esta instrucción se ha cometido un error de sintaxis. Por lo tanto, referente a la forma de escribirlo, no a su significado en el programa.

En el ZX-Spectrum, no parece nunca este mensaje, pero en cambio cuando se entra una instrucción equivocada sintácticamente el sistema de entrada nos lo indica con un interrogante en el lugar donde cree que se ha producido este error sintáctico.

Por otra parte, el error

Subscript wrong

o algo parecido indica que en algún momento se referencia un elemento de una lista o tabla que está más allá de los límites previstos en la instrucción DIM. Evidentemente el BASIC, o el lenguaje que sea, no puede saber más, ya que desconoce nuestra intención al resolver el problema.

Los errores semánticos

Del mismo modo un programa puede ser incorrecto sin necesidad de que se produzca ningún tipo de error. Si en algún lugar de un programa se utiliza la función valor absoluto, en lugar de utilizar la función sacar la raíz cuadrada, el comportamiento del ordenador es en todo instante correcto, pero en cambio, el resultado de ejecutar el programa no es el deseado para resolver el problema.

En resumen, se puede cometer tres tipos de errores:

- Los sintácticos porque escribimos mal lo que el lenguaje requiere.
- Los semánticos que provocan acciones incorrectas en el ordenador o al menos incoherencias entre los datos que manipulamos.
- Finalmente, un último tipo de error que el ordenador no detecta, pero que en definitiva impide que el ordenador resuelva bien el problema.

Al principio de aprender a programar, los errores sintácticos molestan mucho y son los más frecuentes. Después de un poco de experiencia son relativamente fáciles de corregir. Los errores semánticos que el ordenador avisa son ya más difíciles de corregir. Finalmente somos nosotros mismos los únicos que podemos detectar y corregir los errores que se cometen al analizar un problema. Este último tipo de errores es mucho más frecuente de lo que parece; aprenda a reconocerlos.

18.1.2 El modo de ejecución inmediato

El BASIC permite que se disponga de los comandos en modo inmediato; es decir, podemos ejecutar los comandos fuera de un programa.

Esta posibilidad es de gran valor para la depuración de los programas. Aparte de darle una máquina de calcular en sus manos, la ejecución en modo inmediato es una herramienta fundamental para investigar lo que tenemos en memoria cuando estamos ejecutando un programa que nos falla.

Utilice este recurso aun cuando no tenga indicación de dónde está el error. Un repaso de los valores que toman las variables realmente

puede colocarnos en la pista de estos errores que hemos cometido al hacer el análisis del problema.

El seguimiento de un programa

Es muy conveniente que en el desarrollo de un programa tenga un ejemplo calculado a mano de las variables para comprobar que el programa realiza las cosas según está planificado.

Las técnicas de seguimiento, que se han estudiado para seguir los programas, son valiosas.

18.1.3 Los conceptos en la programación en BASIC

A lo largo de la Enciclopedia le hemos enseñado muchas instrucciones del BASIC. Lo hemos hecho de una manera muy detallada y resaltando las diferencias entre cada una de ellas. Es el momento de apreciar las analogías y agrupamientos según el tipo de instrucciones.

18.1.3.1 Los objetos del lenguaje

Las variables y las constantes son los *objetos* básicos sobre los que el lenguaje opera. En ambos objetos se pueden distinguir los números y los textos.

Por otra parte, las *variables* son entidades que evolucionan durante la ejecución del programa; es decir, poseen valores distintos durante el proceso de ejecución.

Las *constantes* en cambio, son entidades que no varían a lo largo de la ejecución del programa.

Adelantando un poco, el concepto de constante y variable aparece en todos los lenguajes de programación. De un lenguaje de programación a otro lo que varía es el tipo de variables y constantes que se pueden usar. En este sentido, el BASIC se puede considerar bastante limitado, pues sólo tiene dos tipos.

De hecho, el concepto de objeto del lenguaje está relacionado con la memoria del ordenador. La memoria física del ordenador no posee estructura, o como máximo una estructura elemental.

Las variables y constantes permiten referirnos a estos trozos de memoria de una manera más estructurada.

18.1.3.2 Las expresiones

Sobre los objetos del lenguaje se realizan operaciones. La manera de expresar estas operaciones y encadenarlas constituyen las *expresiones.*

Los símbolos utilizados para especificar estas operaciones se llaman *operadores.* Los operadores se clasifican según el tipo de operación que realizan.

Tipos de operadores

En BASIC se distinguen cuatro tipos:

a) Aritméticos (Sumar, restar, multiplicar, dividir, potenciar, cambiar de signo o mantener el signo).

b) Textuales (Concatenar textos).

c) Relacionales (Igualdad y desigualdad, mayor, mayor o igual, menor y menor o igual).

d) Lógicos (AND, OR y NOT).

Cuando se escriben expresiones hay que tener en cuenta el orden que se efectúan las operaciones. Para poder expresar este orden se utilizan las operaciones. Para poder expresar este orden se utilizan los paréntesis (el uso de los paréntesis para este fin es casi universal en los lenguajes de programación).

Se dispone de una tabla de precedencia de operaciones para poder escribir expresiones sin necesidad de utilizar un número de paréntesis excesivo.

La precedencia de los operadores es muy general

Esta precedencias suelen estar muy generalizadas a todos los lenguajes. De una manera general, los operadores aritméticos son los de mayor precedencia. Dentro de ellos, el cambio de signo es la de precedencia más alta; le sigue la potenciación, luego la multiplicación y la división y finalmente la suma y la resta. Los operadores textuales suelen estar también al mismo nivel que los aritméticos.

Los operadores relacionales son los que se calculan después, con mayor precedencia para los que indican comparación, tales como el mayor o menor o igual y después los de igualdad y desigualdad.

Los operadores lógicos son los de menor precedencia. Dentro de ellos, la operación NOT es el de precedencia más alta, le sigue el operador AND y finalmente los operadores OR y XOR.

Las expresiones son el núcleo fundamental de la programación y es imposible imaginar un programa práctico sin que se realice algún cálculo, o se utilice alguna expresión.

Esta coincidencia de precedencias en los distintos lenguajes permiten que una vez aprendido cómo construir expresiones en un lenguaje, sea relativamente fácil entender cómo se construyen las expresiones en otro lenguaje. Nos acostumbramos a un estilo.

18.1.3.3 La asignación

Esta operación es la que permite colocar el resultado de una expresión en algún lugar de la memoria. También es una instrucción general a todos los lenguajes. En BASIC se simboliza por la instrucción LET (aunque se puede omitir) y el primer signo igual de una instrucción.

La filosofía fundamental de esta instrucción, que existe en todos los lenguajes de programación, consiste en calcular un valor que corresponde a la parte derecha del signo igual en BASIC; la parte izquierda nos indica en qué zona de memoria se debe colocar este resultado.

18.1.3.4 Las instrucciones de entrada y salida

Son las instrucciones que posee todo lenguaje para comunicarse con los dispositivos periféricos al ordenador propiamente dicho. Estos perifé-

ricos son de la índole más diversa: pantallas, impresoras, teclados, discos, otros ordenadores, etc.

Las instrucciones más particulares

Suelen ser las instrucciones menos generalizables de cualquier lenguaje. En el BASIC las más representativas son el PRINT y el INPUT, pero también deben considerarse instrucciones de este tipo las que manipulan la cinta de cassette (LOAD, SAVE) y las que manipulan los ficheros (OPEN, CLOSE).

Las instrucciones que sirven para manejar los gráficos son maneras especializadas de colocar puntos en la pantalla. Aunque responden también a unas operaciones elementales comunes a todos los sistemas gráficos (PLOT, DRAW, LINE, CIRCLE).

Las instrucciones de sonido son también órdenes de comunicación con los periféricos (SOUND, BEEP).

En definitiva, cualquier elemento que se desea conectar al ordenador requiere un juego de instrucciones para su manejo que dependen mucho del tipo de periférico y, por lo tanto, son instrucciones que se diferencian mucho entre un dialecto del BASIC y otro.

18.1.3.5 Las instrucciones de control

Estas instrucciones son las que designan cómo el lenguaje puede realizar bifurcaciones; es decir, decidir qué cálculos se van a hacer y cuáles no. Estas instrucciones son las que diferencian un ordenador de una máquina de calcular.

Instrucciones básicas de control

Dentro de las instrucciones de control se pueden distinguir tres instrucciones básicas que son necesarias en todo lenguaje de programación. Son las instrucciones de salto.

- La primera es la de salto incondicional. En BASIC, el representante es el GO TO, que significa enviar el programa a otro lugar sin condición alguna; es decir, sin decisión previa.
- La segunda es la de salto condicional que, según una condición, se realizan unas instrucciones o bien unas otras. En BASIC el representante de esta instrucción es el IF ... THEN .... En este tipo de instrucción se pregunta por una condición antes de establecer una bifurcación en las operaciones a realizar.
- La tercera es una llamada a una subrutina; es decir, la realización de un salto incondicional con memorización del lugar desde donde se ha realizado. De esta manera se puede volver al lugar de la llamada cuando deseamos. En BASIC se escribe como GO SUB.

Estas tres instrucciones son las más elementales y se encuentran ya en las instrucciones de código máquina que tiene la unidad central de procesos.

En el BASIC, como en todos los lenguajes, existen formas más complicadas, pero que conceptualmente se corresponden con esquemas más o menos complicados de las instrucciones elementales.

En BASIC la instrucción de control más complicada es el bucle FOR/NEXT. Ya sabe, sin embargo, que esta instrucción no es estrictamente necesaria. Se puede simular perfectamente con el IF ... THEN y el GO TO.

Estos esquemas complicados que generan otro tipo de instrucciones no son necesarios, pero la estructura de la decisión es tan frecuente que la necesidad de la instrucción queda plenamente justificada.

En los dialectos de BASIC modernos han aparecido una serie de construcciones que se denominan estructuradas porque pretenden seguir las indicaciones que hace la programación estructurada. Instrucciones WHILE (no se ha visto en este curso) para los bucles mientras, las instrucciones ON ... GO TO y ON ... GO SUB, la instrucción SELECT y CASE (no se ha visto en este Curso) para los casos de alternativa múltiple. En definitiva cada una de ellas se puede simular con sólo las instrucciones GO TO y IF ... THEN usados con cierta disciplina.

No olvide que la herramienta más potente para estructurar un programa es la subrutina, ya que permite centrar la atención sobre el qué se hace; es decir, en la función de más que en el cómo se hace, que corresponde al contenido de la subrutina.

18.1.3.6 Los datos estructurales

Se entienden como datos estructurales aquellas construcciones que permiten referirnos a un conjunto de datos elementales de una manera única o sistemática.

El BASIC dispone de las tablas como elementos de datos estructurados. Permite manejar varias variables de un mismo tipo mediante la utilización de los subíndices. La instrucción DIM complementa esta estructura para efectuar la reserva de memoria adecuada.

Existen lenguajes en que la riqueza de estructuras es prácticamente infinita. En este sentido el BASIC es muy limitado.

La tabla es la estructura de datos más necesaria

Hay que reconocer que a pesar de esta limitación incorpora la estructura de datos más útil y más necesaria. De la misma manera que las instrucciones de salto condicional, incondicional y a subrutina constituyen las acciones elementales de control de flujo, las tablas son el elemento básico para las estructuraciones de datos más complicadas. Volvemos al binomio necesidad y comodidad.

Finalmente, hay que advertir que los textos, aunque le parezcan algo muy natural dentro del ambiente del BASIC no lo son tanto en otros lenguajes.

Los textos en el fondo son tablas de caracteres; por lo tanto, corresponden a una estructura de datos y no a un dato elemental.

Este aspecto hay que tenerlo muy en cuanta y desde el punto de lo que significa el programa que nos permite utilizar el BASIC es el aspecto más complicado. Observe que no es nada evidente cómo se administra la memoria, cuando un texto se alarga o acorta (se utiliza con frecuencia la concatenación de textos). Una mala gestión nos produce una pérdida de memoria notable y una buena administración posiblemente una notable pérdida de tiempo.

En realidad es la parte del BASIC que parece más simple al usuario, pero es la más complicada para el programador que realiza el programa interpretador del BASIC.

Si algún día programa en otro lenguaje de programación, el tratamiento de los textos será lo que más en falta encontrará a faltar en el BASIC.

18.1.3.7 Las funciones

Las funciones son un tipo especial de subrutinas que nos devuelven un valor. El BASIC dispone de esta herramienta. Es interesante el concepto de los argumentos dentro de las funciones. En muchos otros lenguajes se utilizan las subrutinas como las funciones, es decir, con argumentos.

En este punto conviene comentar que el BASIC dispone de muchas funciones intrínsecas; es decir, están incorporadas como elementos sintácticos del propio lenguaje. Este repertorio puede ser muy amplio o muy reducido, dependiendo del dialecto de BASIC que se trate.

En definitiva, lo que se pretende es que el programador encuentre una serie de operaciones frecuentes introducidas en el lenguaje y, además y más importante, poder realizar estas operaciones a nivel de lenguaje máquina, con lo que la eficiencia de la operación mejora grandemente.

Las funciones se asocian en cuanto a mecanismo de funcionamiento con las subrutinas. Sin embargo, deben entenderse, al menos dentro del contexto del BASIC, como objetos diferentes con objetivos diferentes.

La diferencia principal que hay entre una subrutina y una función en BASIC es que la función sólo admite una expresión y la subrutina utiliza varias instrucciones, que pueden incluir, además de expresiones, otros tipos de instrucciones.

Una función en BASIC permite resumir mediante una llamada la evaluación de una expresión que se usa frecuentemente.

Las funciones son semánticamente equivalentes a los operadores

Una función es un mecanismo de extensión de los operadores. A primera vista parece imposible que una función se parezca a un operador. Sin embargo, es bastante sencillo imaginar un lenguaje en el que no existen los operadores.

Tomemos como ejemplo la siguiente expresión en BASIC:

```
LET A = 3+4*5 ,
```

es fácil escribir esta expresión como

```
LET A = FNSUM(3,FNMUL(4,5))
```

y que se han definido las funciones FNSUM y FNMUL como las siguientes:

```
10 DEF FNSUM(X,Y) = X+Y
20 DEF FNMUL(X,Y) = X*Y
```

La expresión se ha escrito sin necesidad de ningún operador y se ha resuelto mediante las llamadas a una función. Estas funciones podrían ser intrínsecas.

Es posible diseñar un lenguaje sin operadores pero con funciones. Esta correspondencia entre operador y función permite saber de qué tipo de operador se trata, si unario, binario, ternario, etc.

El operador unario se corresponde con una llamada a una función que sólo utiliza un argumento. Por ejemplo, el operador cambio de signo se puede asociar a una función del tipo

```
10 DEF FNCMS(X) = -X
```

que sólo requiere un argumento.

Las funciones que simulan los operadores de suma y multiplicación mencionadas más arriba se relacionan con los operadores de tipo binario porque requieren dos argumentos. De la misma manera la función intrínseca LEFT$(A$,N) se asocia con un operador binario.

En muchos dialectos del BASIC se corresponde con el operador de fragmentación. Por ejemplo se puede utilizar en muchos de ellos una cosa parecida a A$[1:N], o bien como en el ZX-Spectrum que se utiliza A$(1 TO N).

La función MID$(A$,N,M) se corresponde con la idea de operador ternario pues requiere tres argumentos. Los BASIC que disponen de operadores de fragmentación expresan esta función de un modo parecido a A$[N:N+M−1], o bien A$(N TO N+M−1).

Por lo tanto, las funciones tienen el significado de operador, y el tipo de operador se relaciona con el número de argumentos que precisa la función para evaluarse.

La figura 1 resume estas características generales para el lenguaje BASIC.

18.2 INTRODUCCIÓN A LOS LENGUAJES DE PROGRAMACIÓN

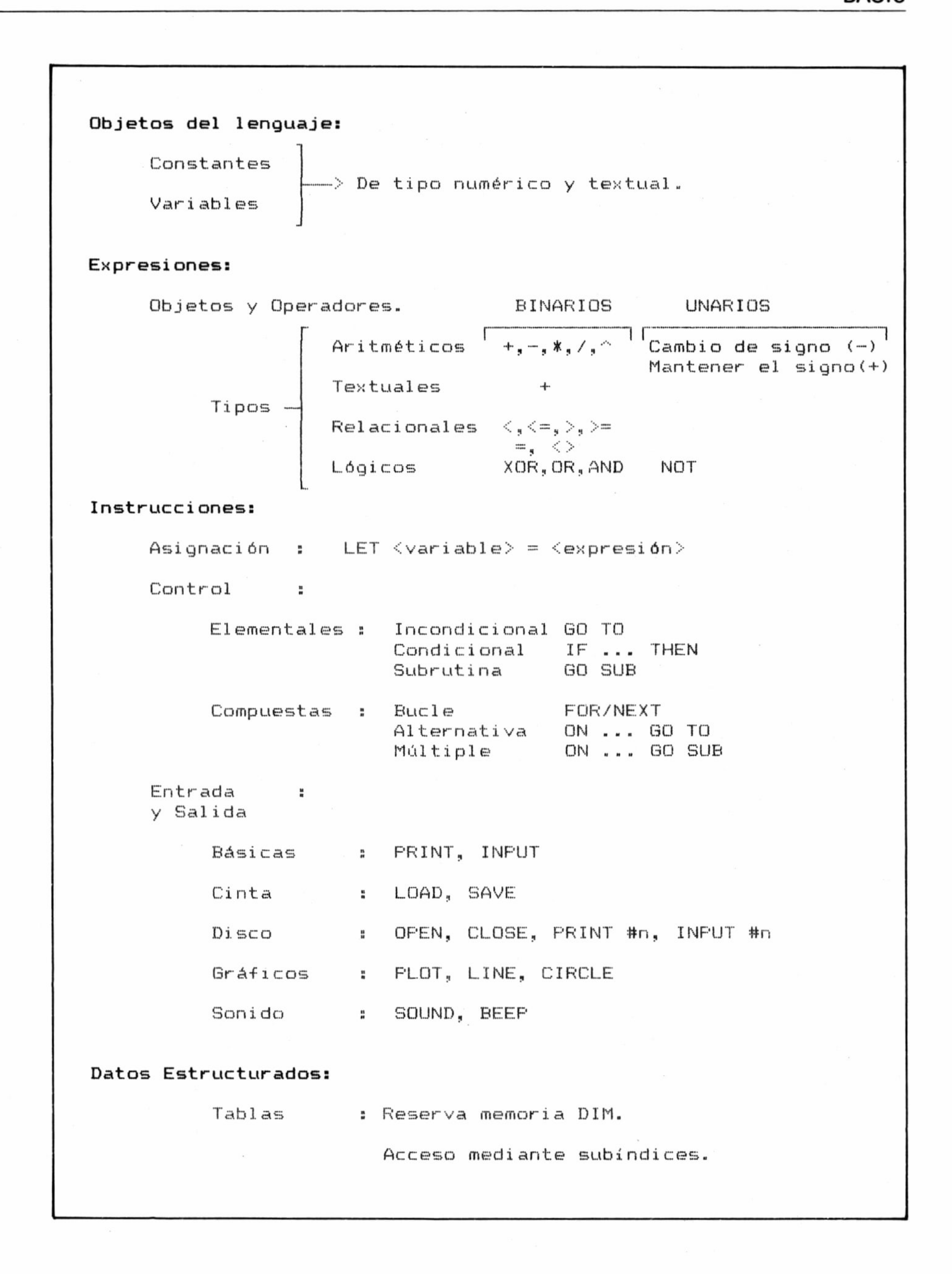
El lenguaje BASIC ha servido para introducirnos en el mundo de los ordenadores. La misión del lenguaje ha sido facilitarnos la comunicación con el verdadero corazón de la máquina, la unidad central de procesos (en castellano abreviada UCP y en inglés CPU).

En este apartado vamos a fijar más nuestra atención en cómo funciona un lenguaje que en los aspectos de cómo se escribe. Este segundo aspecto lo consideraremos en el apartado siguiente. Es decir, consideraremos los aspectos semánticos generales, antes que los sintácticos.

18.2.1 Lenguajes máquina y ensamblador (assembler)

Como se ha visto en el capítulo 17, la CPU funciona mediante un código numérico, cada número representa una operación distinta de las que realiza la unidad de control de procesos.

Los elementos esenciales de la CPU son el contador de programa que nos indica dónde se encuentra la nueva instrucción a realizar. El



```
Objetos del lenguaje:

     Constantes
                 ---> De tipo numérico y textual.
     Variables

Expresiones:

     Objetos y Operadores.                BINARIOS        UNARIOS

                  Aritméticos     +,-,*,/,^      Cambio de signo (-)
                                                 Mantener el signo(+)
                  Textuales          +
          Tipos --
                  Relacionales    <,<=,>,>=
                                   =, <>
                  Lógicos         XOR,OR,AND     NOT
Instrucciones:

     Asignación   :   LET <variable> = <expresión>

     Control      :

          Elementales :  Incondicional  GO TO
                         Condicional    IF ... THEN
                         Subrutina      GO SUB

          Compuestas  :  Bucle          FOR/NEXT
                         Alternativa    ON ... GO TO
                         Múltiple       ON ... GO SUB

     Entrada      :
     y Salida

          Básicas     :  PRINT, INPUT

          Cinta       :  LOAD, SAVE

          Disco       :  OPEN, CLOSE, PRINT #n, INPUT #n

          Gráficos    :  PLOT, LINE, CIRCLE

          Sonido      :  SOUND, BEEP

Datos Estructurados:

          Tablas      : Reserva memoria DIM.

                        Acceso mediante subíndices.
```

Figura 1. Síntesis del BASIC

estado de la CPU oscila entre dos estados, lectura de la instrucción contenida en la posición de memoria, que indica el contador, y ejecución de la instrucción.

El incremento del contador permite ir realizando las instrucciones contenidas en posiciones sucesivas de memoria. Tenga en cuenta que este incremento del contador puede ser alterado por la propia instrucción. Este mecanismo es el equivalente a la instrucción GO TO del BASIC en el lenguaje máquina. El equivalente al contador de la CPU en lenguaje BASIC son los números de línea; se ejecutan uno detrás de otro, a menos que se altere mediante un GO TO.

Un programa (en cualquier lenguaje cuando se ejecuta) es una secuencia de números con significado para la CPU. Se supone que esta secuencia de números expresa la acción global que deseamos que realice el citado programa.

Dése cuenta de que cualquier programa en último término se debe expresar en forma de lenguaje máquina, porque es el único lenguaje que entiende la CPU.

La frase en lenguaje máquina

Esta secuencia de números, el programa, se puede considerar como una *frase* construida con un alfabeto cuyas letras.son los código numéricos. En este sentido es como debe entenderse el término aplicado el código numérico de la CPU. Observe que en este lenguaje no existe un equivalente de lo que es sílaba y palabra del lenguaje natural.

El lenguaje ensamblador o assembler no es más que una mejora relativamente pequeña de esta situación. Se sustituye el código numérico por símbolos (tal como se ha indicado en el capítulo 17) que permiten recordar más fácilmente cuáles son las distintas operaciones de la CPU.

Esta facilidad se introduce a costa de tener que utilizar un programa en lenguaje máquina que permita realizar de una manera automática la traducción de estos símbolos a los códigos que requiere la CPU.

Las características siguientes son las que distinguen a estos dos lenguajes de los demás lenguajes de programación.

a) El nivel de lenguaje: una instrucción en lenguaje de máquina desencadena una acción mucho más elemental y primitiva que una instrucción en un lenguaje como el BASIC. Cuando en BASIC se escribe una instrucción PRINT para visualizar una variable textual, a nivel del lenguaje máquina o assembler esta instrucción es impensable. Se dispone como mucho de una instrucción que visualiza un carácter. Por lo tanto, para visualizar un texto es necesario programar un bucle y un movimiento de textos que requiere bastantes más instrucciones. No piense que la situación mejora si la variable es numérica; antes al contario, se requiere para visualizarla una transformación a una cadena de caracteres que no es nada sencilla; algo así como una instrucción STR$, pero programada a nivel del lenguaje de máquina.

b) Falta de estructura. El programa entero no posee absolutamente ninguna estructura. Para estructurar es necesario realizar un gran esfuerzo de documentación, para resaltar los puntos estructurales importantes que no posee el lenguaje en sí mismo. Se consigue a base de mucha disciplina en la manera de escribir los programas, sin que este esfuerzo rinda demasiados beneficios.

c) Errores de escritura. El número de errores que se cometen en general no depende del lenguaje sino del número de instrucciones escritas. Es evidente que para realizar un programa en lenguaje máquina es necesario escribir muchas más instrucciones que en lenguaje BASIC y la proporción de errores es la misma, con lo que en términos absolutos cometeremos muchos más errores en un programa escrito en lenguaje máquina.

18.2.2 Lenguajes interpretados y compilados

Según la manera como se ejecuta un programa se clasifica en compilado o interpretado. En último término, todo lenguaje se traduce a lenguaje máquina. El criterio de ejecución que utilizamos en esta clasificación es en relación a la manera de escribirlo.

El título de este apartado se puede entender equivocadamente. De hecho, la propiedad de compilado o interpretado no es del lenguaje sino de la forma de ejecución del programa que nos permite utilizar un determinado lenguaje. Sin embargo, lo utilizamos así porque muchas veces al leer temas informáticos esta propiedad se asocia a un lenguaje por abuso de expresión.

Por ejemplo, el BASIC se puede presentar de las dos formas, tanto interpretado como compilado. En otros lenguajes como los ensambladores y el código máquina sólo se consideran compilados.

Para entender bien este criterio de clasificación profundicemos en el mecanismo de funcionamiento del BASIC.

Cuando escribimos un programa en BASIC utilizamos palabras propias del lenguaje para describir la manera de resolver el problema que queremos que resuelva el ordenador.

Ciclo del funcionamiento básico en un lenguaje interpretado

En el momento de ejecutarse el programa el BASIC realiza dos pasos:

a) Traducción: El texto que constituye la instrucción escrita en BASIC se analiza y se traduce el código máquina equivalente para realizar el significado de la instrucción. Generalmente se hace mediante una traducción a llamadas a subrutinas escritas en código máquina.

b) Ejecución: El código máquina obtenido en la fase anterior se ejecuta sobre la CPU.

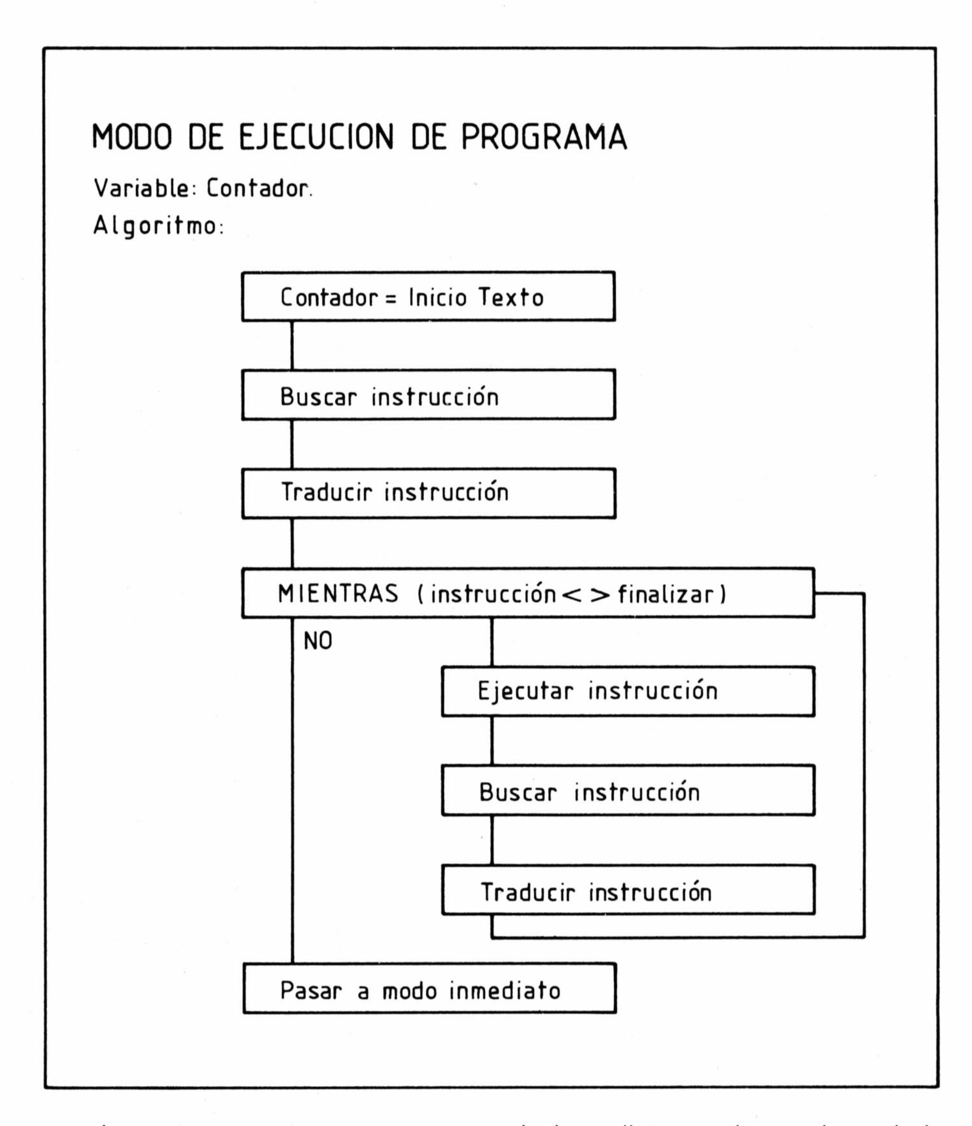
En la figura 2 se muestra el ordinograma de funcionamiento del BASIC en modo de programa. Cuando tecleamos el RUN, colocamos el contador del programa BASIC a la primera línea del texto. Se inicia entonces la búsqueda de la primera instrucción, se traduce y se ejecuta, se busca la siguiente y se repite la traducción y ejecución, en tanto que la instrucción alcanzada sea diferente de finalizar, es decir, un STOP, un END. (Este esquema es simplificado, pues se puede finalizar cuando se toca la tecla de interrupción).

En la figura 3 se muestra el ordinograma de funcionamiento del BASIC en modo inmediato. Se trata de un bucle sin fin que lee instrucciones del teclado, las traduce y las ejecuta. Sólo se finaliza desconectando el ordenador. En ordenadores que poseen diskettes probablemente existe un modo de finalizar que permite retornar al sistema operativo.

En el fondo, la ejecución en modo inmediato y con el RUN sólo se distinguen por quien da la orden de ejecución de una instrucción. En el caso de ejecución en modo inmediato la damos nosotros mismos, mientras que cuando estamos en forma de RUN la finalización de una línea de programa o encontrar dos puntos son los desencadenantes de la repetición del ciclo.

Es aproximadamente correcto considerar que un programa puede realizarse con instrucciones ejecutadas en modo inmediato, entrando una detrás de otras las instrucciones del programa. Decimos que es aproxi-

Figura 2. Esquema de ejecución del BASIC en modo de programa.



madamente correcto, porque en modo inmediato no tienen demasiado sentido las instrucciones GO TO. Somos nosotros que decidimos (quizá con la colaboración de un resultado del programa) cuál es la siguiente instrucción a ejecutar.

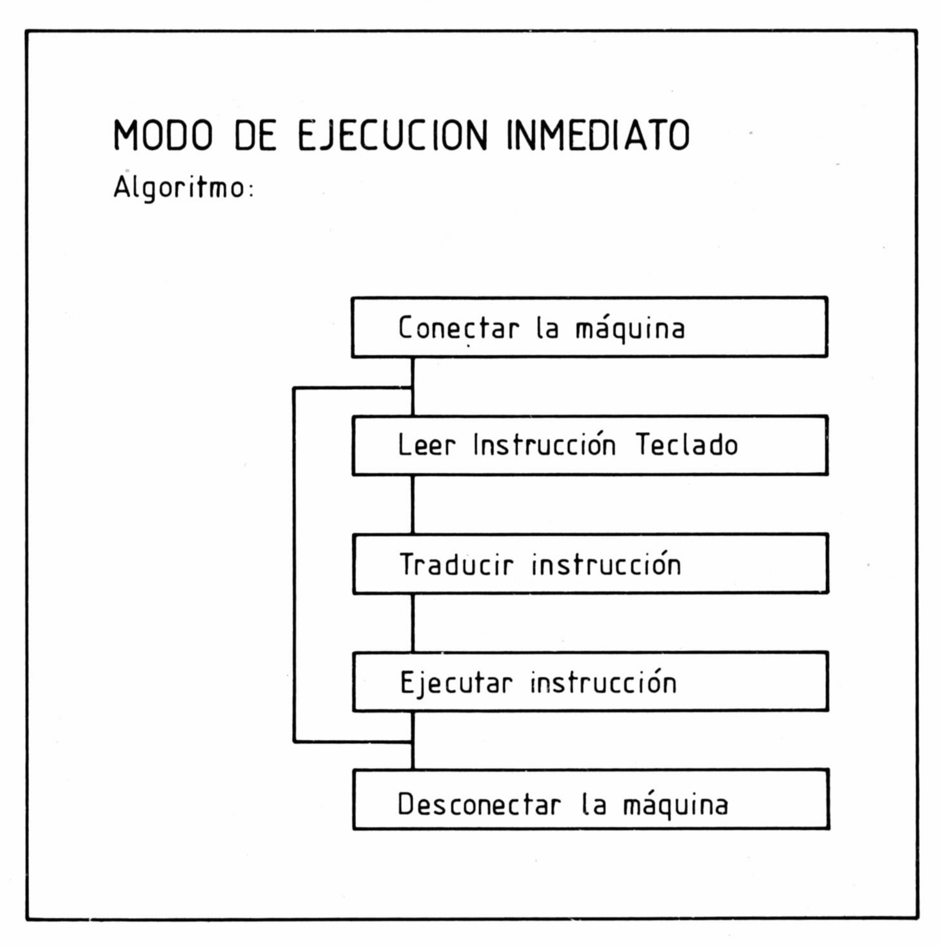
El análisis de esta manera de funcionar sugiere un modo alternativo ¿no es posible realizar la fase de traducción primero y luego realizar la ejecución del programa? La respuesta es sí. Justamente esta es la idea que se aplica a los lenguajes compilados. Sustituyen estas dos fases en al ejecución de las instrucciones que tiene el BASIC por otro esquema de elaboración en el tiempo.

Fases de desarrollo en lenguaje compilado

El desarrollo y ejecución de un programa en un lenguaje compilado consta de las fases siguientes:

a) Escritura de un programa como un texto cualquiera. La figura 4 muestra esta fase. Desde el teclado y con un programa llamado *Editor* se prepara el texto del programa igual que en una máquina de escribir.

Figura 3. Esquema de ejecución del BASIC en modo inmediato.



El resultado se almacena en un fichero texto, es decir, que contiene el programa en forma de texto normal. (Las máquinas que utilizan lenguajes compilados utilizan los diskettes, de lo contrario el proceso sería sino imposible, sí muy engorroso.)

b) Traducción del texto al lenguaje máquina. La figura 4 muestra también esta fase. Se utiliza el fichero texto, de la fase anterior, como entrada de un programa (es decir, toma el texto como dato) traductor que se encarga de transformar en una sola vez el texto escrito en el código de máquina que representa. Este programa se denomina *Compilador.* Esta traducción se almacena en un fichero llamado objeto, ya que está precisamente en código de máquina.

c) Montador. Como un programa se puede componer de varios módulos que constituyen las bibliotecas de módulos, es necesario agruparlos todos juntos en único módulo ya listo para ejecutar. La figura 5 muestra un esquema del este proceso. Al programa que realiza la operación de montaje se le llama MONTADOR (en inglés LINK o LINKER).

d) Ejecutar el programa en código máquina. El programa traducido se carga en la memoria del ordenador y se inicia su ejecución. La figura 6 muestra esta fase, al programa que realiza esta tarea se le denomina

Figura 4. Fase de compilación de un programa.

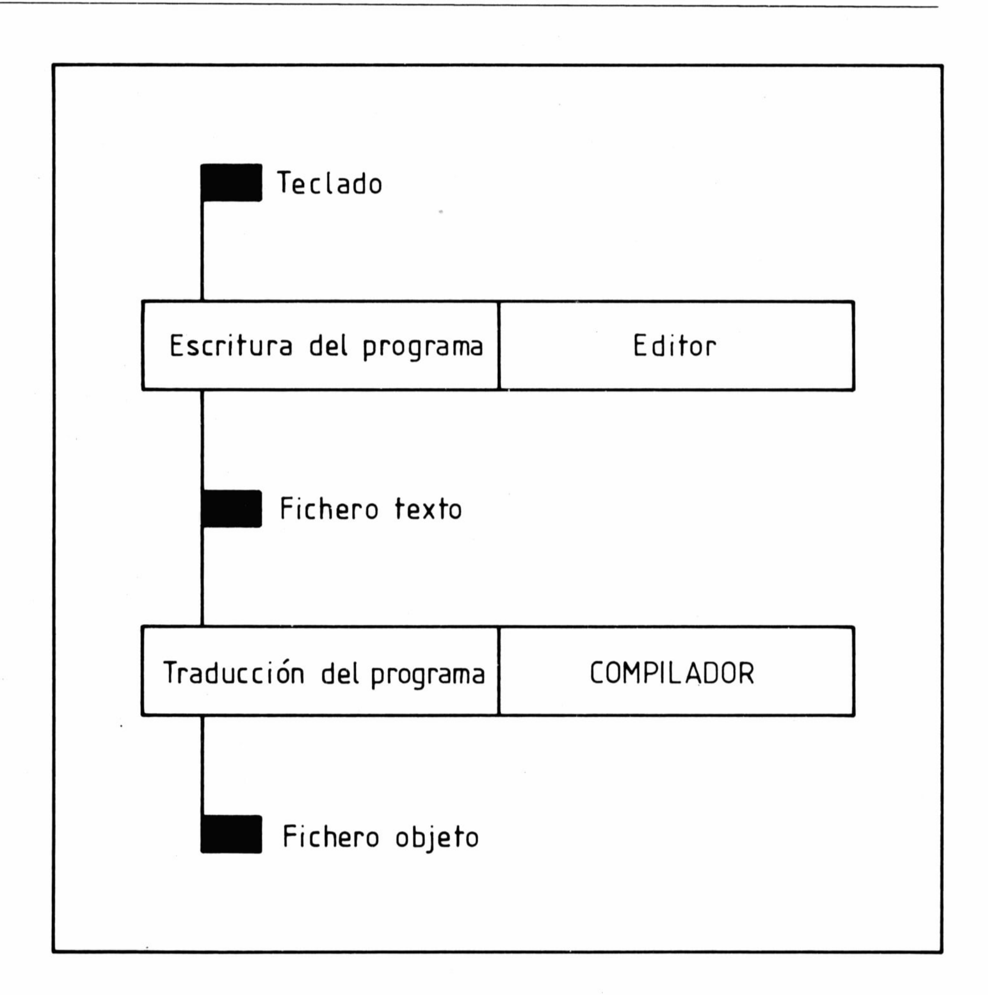


Figura 5. Fase de montaje de un programa a partir de módulos compilados.

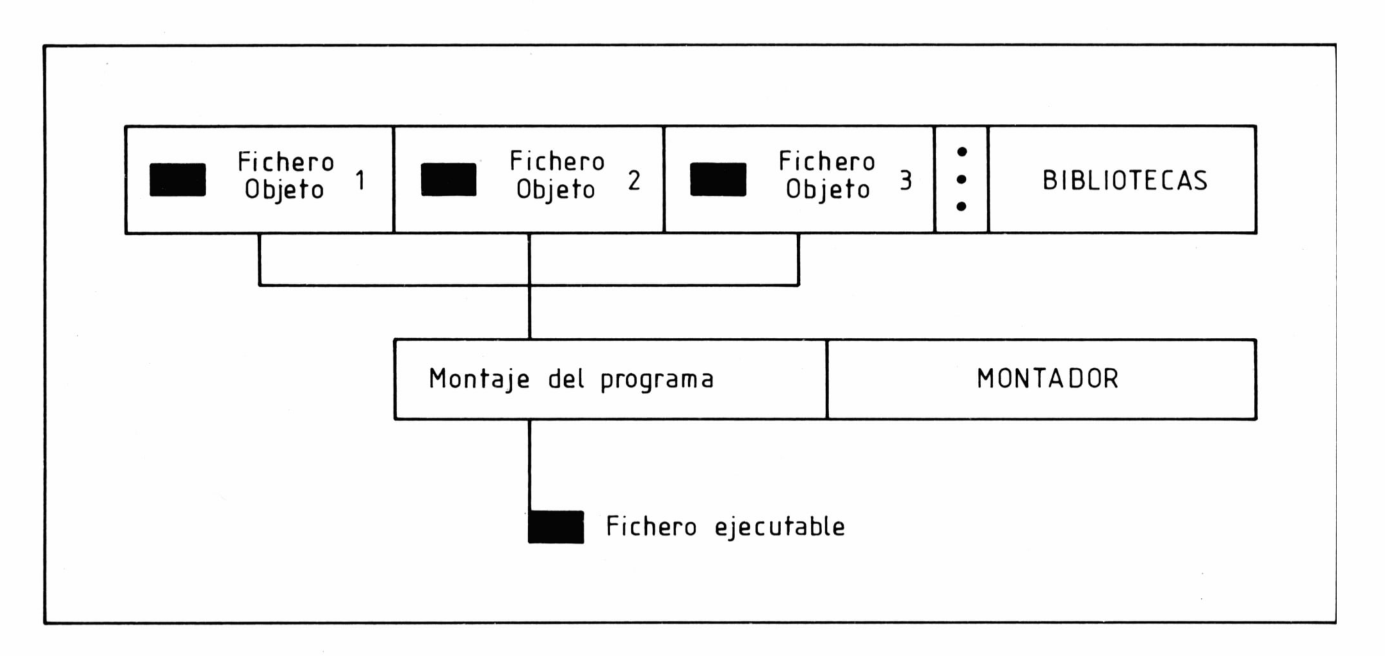
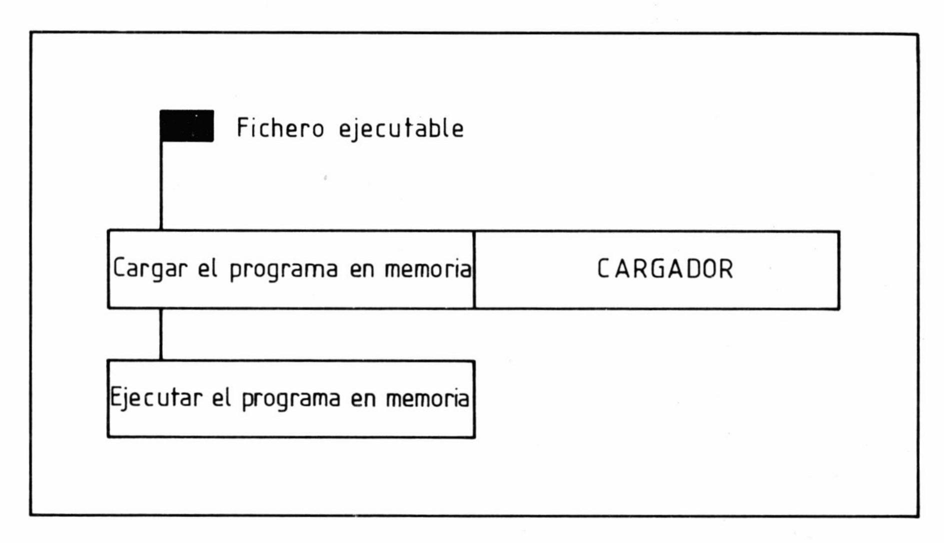


Figura 6. Fase de ejecución de un programa compilado.



Cargador. Naturalmente este proceso puede repetirse tantas veces como se quiera, sin necesidad de volver a realizar las fases anteriores.

Por lo tanto, los lenguajes que siguen un esquema de traducción como el BASIC se denominan *interpretados*, ya que la traducción es simultánea con la ejecución. Los que siguen este segundo esquema se denominan *compilados* porque la traducción (compilación del programa) está separada de la fase de ejecución.

Las consecuencias de utilizar un esquema de ejecución compilado en comparación a uno interpretado son:

Comparación entre un lenguaje compilado e interpretado

a) El tiempo de ejecución decrece enormemente. Si un programa en BASIC tiene una duración de 100 unidades de tiempo, el mismo programa escrito en un lenguaje compilado (el BASIC también puede compilarse) tarde en ejecutarse de 1 a 30 unidades de tiempo. Esta relación es muy variable y depende de tres factores, del compilador utilizado, el nivel del lenguaje y del programa a traducir.

En primer lugar , *depende del compilador*, porque el esquema de traducción no es único necesariamente y, por lo tanto, un programa traductor puede generar un código máquina más eficiente que otro.

En segundo lugar, *depende del nivel del lenguaje.* Cuando el lenguaje es de muy bajo nivel, es decir, cercano al assembler los objetos y las instrucciones son más elementales. Esto permite programar con más eficiencia algunos casos particulares que se dan en nuestro problema. En un lenguaje de alto nivel es más difícil tratar estos casos particulares. Por ejemplo, en el caso del BASIC las variables numéricas son todas reales, con decimales, (excepto en algunos tipos de dialectos). Si se utiliza otro tipo de lenguaje donde existen variables de tipo entero, sin decimales, el programa que resulta será más eficiente si los datos a manipular son todos enteros, ya que el tipo de datos del problema se acerca más a los objetos básicos que manipula el lenguaje.

En tercer lugar, *depende del programa a realizar.* Por ejemplo, si un programa lee muchos datos de una cinta de cassette es obivio que va

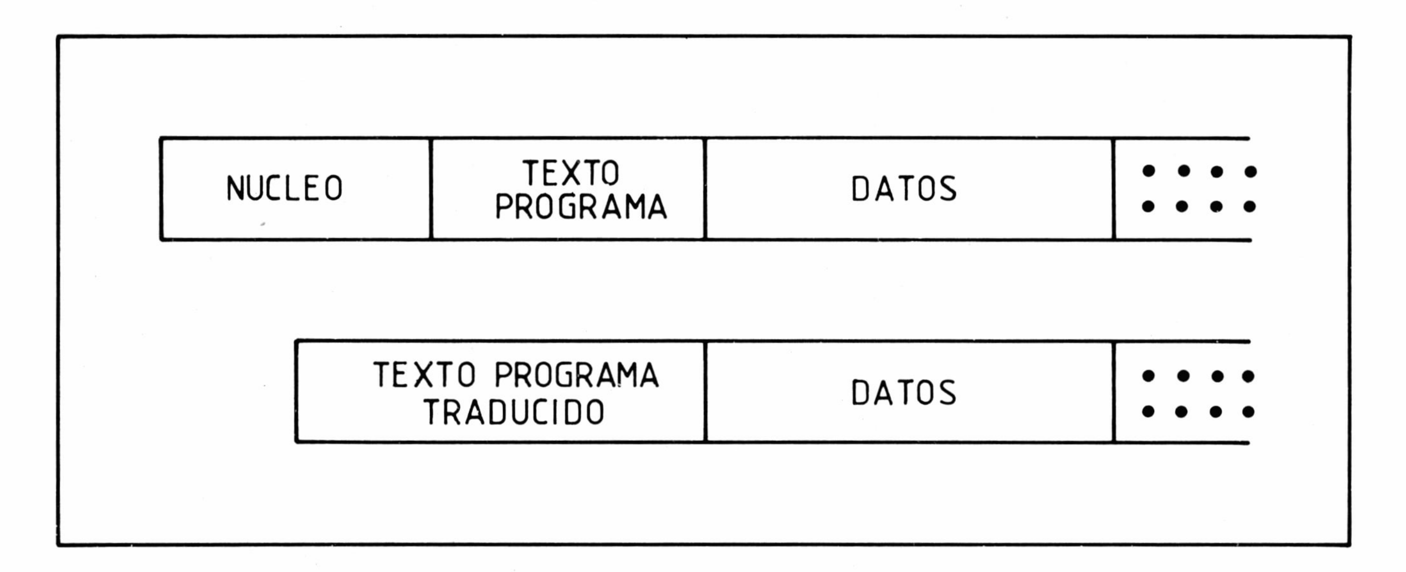


Figura 7. Comparación de la distribución de la memoria del mismo programa interpretado y compilado.

importar poco cuánto tarde el programa. La operación más lenta será la de leer el cassette y no importará si se tarda mucho o poco en realizar las instrucciones del programa. En cambio, para un programa de gráficos el factor de tiempo es imprescindible. Recuerde la representación de funciones en gráficos de tres dimensiones en el capítulo 15, en el tomo anterior.

b) No existe el modo inmediato de ejecución. No es posible ver el estado en que están las variables ni hacer cálculos con las instrucciones del lenguaje. El mecanismo de traducción no está en el mismo programa. Al haberse realizado la traducción previamente, las instrucciones de traducción no se incorporan en nuestro programa.

Este punto es lo que hace atractivo el BASIC para iniciarse en la programación ya que todo el mecanismo de cálculo e inspección de las variables es tan sencillo como el propio lenguaje de programación.

En los lenguajes compilados esta faceta se complica, ya que en realidad el programa que se ejecuta está en lenguaje máquina y hay que realizar inspecciones al mismo nivel; es decir, en lenguaje máquina. Esto requiere unos conocimientos específicos de la máquina elevado.

c) Reducción de memoria. Se consigue una reducción de la memoria utilizada por el programa. Esta reducción se da en dos aspectos: en primer lugar, no debe incorporarse el programa traductor y, en segundo lugar, las instrucciones que se han escrito se almacenan traducidas. Por lo tanto, no es necesario mantener el texto del programa que hemos escrito. La figura 7 muestra un esquema comparativo de la utilización de la memoria en ambos modos.

La reducción de memoria es importante en el primer aspecto. En la mayoría de ordenadores personales el programa traductor de BASIC, el NÚCLEO en la figura, se incorpora en el ordenador en la memoria de sólo lectura (ROM) para que este programa sea idestructible por el usuario. Es decir, no es posible utilizar esta memoria para otro programa. Pero esto no sucede en máquinas profesionales. La memoria que ocupa un traductor de BASIC es desde 16KB hasta 64KB, dependiendo del tipo de

dialecto. En una máquina que es posible aprovechar la memoria es una reducción importante.

En el segundo aspecto, la reducción de memoria es pequeña, el texto de un programa como tal ocupa 4 o 5 KB en los más largos, aunque este número puede aumentar. En términos porcentuales esta reducción de memoria se sitúa sobre el 40 o 50 %.

Observe que en la figura el programa traducido requiere una parte del NÚCLEO ya que el programa requiere alguna de las funciones que contiene el NÚCLEO. Además del traductor, el NÚCLEO contiene ciertas funciones de cálculo.

En términos globales, es decir, contando el núcleo más el texto en el programa interpretado y el programa traducido se obtiene una reducción de alrededor del 30 %, aunque este número es también variable.

d) Bibliotecas. Los lenguajes compilados se prestan mejor a la construcción de bibliotecas. Estas bibliotecas son trozos de lenguajes traducidos. Los nombres de las variables se han transformado en direcciones de la memoria, que administra cada trozo.

En BASIC es posible también realizar bibliotecas a base de almacenar textos, pero cuando se utilizan, pueden haber conflictos con los nombres de las variables de los distintos módulos. En los lenguajes compilados estos conflictos desaparecen ya que los nombres de las variables se relacionan con las direcciones de memoria del propio módulo. En otro módulo, al mismo nombre le corresponde una posición de memoria distinta.

18.2.3 Ventajas e inconvenientes

En la valoración de estas fierencias la reducción de tiempo y memoria suelen ser las más importantes cuando lo que interesa es la explotación de un programa. Cuando se ha realizado un programa que calcula facturas, lo que interesa es que haga lo más rápidamente las facturas. En general, no interesa estudiar los estados en que se encuentran las variables ni cosas por el estilo. En este sentido un programa compilado tiene una prestación superior que un programa interpretado.

En el momento de desarrollar un programa, es decir, concebirlo, diseñarlo y probarlo, la posibilidad de ejecución en modo inmediato son muy apreciadas. Sin embargo, se pueden compensar bastante bien con buenas bibliotecas.

Se puede pensar que el ideal consiste en desarrollar un programa en lenguaje interpretado y cuando ya funciona se compila. De hecho esta política es la que se sigue cada día más en los desarrollos profesionales. Sin embargo, los actuales compiladores no son demasiado eficientes (comparados con los lenguajes estrictamente compilados) y en la informática profesional aún se prefieren los lenguajes estrictamente compilados, porque además es posible utilizar un tipo de datos que se acerca mucho más al tipo de memoria que dispone el ordenador.

Concepto de máquina virtual

Un lenguaje interpretado se puede imaginar también como una *máquina virtual* que significa que el usuario utiliza una máquina que no existe en realidad como circuitos, sino que está construida mediante programas

o funciones que nos dan una imagen del ordenador mucho más elaborada de lo que es en realidad. El BASIC es un ejemplo típico de esta imagen. Hemos podido programar sin necesidad de hacer ninguna referencia a las propiedades del microprocesador de su ordenador; es más, con diversas máquinas y con diversas organizaciones el BASIC se presenta de la misma manera.

Este concepto de máquina virtual es relativamente moderno y muy útil ya que permite prescindir de las características del ordenador y centrarnos en las características del problema. Muchos de los programas de aplicación se diseñan según esta filosofía. El efecto de esta concepción permiten acercar al usuario al ordenador rápidamente, graduando los esfuerzos para llegar a comprender el mecanismo de funcionamiento de los ordenadores, o en todo caso sacarles provecho.

Otra vez se aplica el principio de: sabemos *qué* hace, pero no *cómo* lo hace, y en consecuencia cómo utilizarlo.

RESUMEN

La *sintaxis* es aquella parte del lenguaje que indica cómo hay que escribir. La *semántica* indica cuál es el significado de lo que se escribe.

En la programación se distinguen tres tipos de errores, los *sintácticos* que provienen de escribir mal una instrucción. Los *semánticos* que detecta el lenguaje debido a incoherencias internas y finalmente los que cometemos nosotros mismos, que el lenguaje no detecta.

Los objetos del lenguaje de programación del BASIC son las *variables* y las *constantes.* Estas a su vez son de dos tipos *numéricas* y *textuales.*

Las expresiones se realizan con los *objetos* y los *operadores.* Los operadores se clasifican en *aritméticos*, *textuales*, *relacionales* y *lógicos.* El orden de enumeración utilizado es de *precedencia decreciente.*

La *asignación* es la instrucción para cambiar el valor de las variables.

La comunicación con un periférico cualquiera requiere utilizar instrucciones de entrada y salida.

Las *instrucciones de control* permiten hacer la decisión de realizar unos cálculos y no otros. Son tres fundamentales son el *salto incondicional*, el *salto condicional* y la *llamada a la subrutina.*

La agrupación de datos que permite el BASIC son las *tablas.*

Las funciones en BASIC se pueden considerar como operadores. Hay una equivalencia entre función y operador.

Una secuencia de números con significado a la CPU constituyen una *frase* en el lenguaje de máquina. Este lenguaje es el de más bajo nivel, el menos estructurado y en el que se comenten más errores de escritura.

En un lenguaje interpretado, el lenguaje realiza siempre la traducción de una instrucción a lenguaje máquina y posteriormente la ejecuta.

En un lenguaje compilado se traduce primero el programa a lenguaje máquina entero y se ejecuta en código máquina.

Cuatro son las fases que deben realizarse para poder ejecutar un programa compilado.

a) Escritura con el programa EDITOR.

b) Traducción con el programa COMPILADOR.

c) Montaje con el programa MONTADOR (LINK).

d) Ejecución con el programa CARGADOR.

Las diferencias de utilizar un compilador respecto a un programa intérprete son:

a) Decrece el tiempo.

b) Desaparece el modo de ejecución inmediato.

c) Se reduce la memoria ocupada.

d) Se pueden realizar bibliotecas de módulos.

Los programas interpretados dan unas propiedades aparentes al ordenador que en realidad provienen de los programas. Esta máquina se denomina virtual ya que se ha realizado con programas.

EJERCICIOS DE AUTOCOMPROBACIÓN

Completar las frases siguientes:

1. La .... es la parte del lenguaje que nos indica cómo hay que escribir las cosas.

2. La .... es lo que nos indica cuál es el significado de las cosas que escribimos.

3. Los errores que se cometen al programar son de tres tipos: sintácticos, ..., detectados por el programa y semánticos no detectados por el programa.

4. Los objetos del lenguaje BASIC son las constantes y las ....

5. Las .... son el encadenamiento ordenado de objetos del lenguaje y operadores.

6. Los operadores relacionales son de .... precedencia que los aritméticos.

7. Las instrucciones básicas de control son el GO TO, el .... y el GO SUB.

8. Un operador se puede sustituir por la llamada a una ....

9. Un .... es un programa que traduce un lenguaje de programación a lenguaje de máquina.

10. Un lenguaje compilado .... el tiempo de ejecución de un programa interpretado.

Rodee con un círculo la letra que corresponda a la respuesta correcta.

11. Cuando obtenemos un error que nos indica que un subíndice de una tabla está fuera de los márgenes previstos, cometemos un error de tipo:

    a) Sintáctico.

    d) Semántico.

    c) De diseño.

    d) De hardware.

12. La posibilidad de realizar una ejecución de modo inmediato se debe a que un programa es:

    a) Compilado.

    b) Estructurado.

    c) Sin estructura.

    d) Interpretado.

13. Los operadores de menor precedencia son:

    a) Relacionales.
    b) Textuales.
    c) Lógicos.
    d) Aritméticos.

14. Los datos estructurados en BASIC son:

    a) Los reales.
    b) Las instrucciones.
    c) Las tablas.
    d) Los operadores.

15. Un operador que se puede sustituir por una llamada a una función de dos argumentos, es equivalente a un operador de tipo:

    a) Unario.
    b) Binario.
    c) Ternario.
    d) Cuaternario.

16. El programa que se encarga de cargar un programa compilado en memoria y ejecutarlo es el:

    a) EDITOR.
    b) COMPILADOR.
    c) MONTADOR.
    d) CARGADOR.

17. El programa que se encarga de preparar el texto de un programa es el:

    a) EDITOR.
    b) COMPILADOR.
    c) MONTADOR.
    d) CARGADOR.

18. El programa que se encarga de realizar las conexiones entre distintos programas ya traducidos al lenguaje máquina es el:

    a) EDITOR.

    b) COMPILADOR.

    c) MONTADOR.

    d) CARGADOR.

19. La reducción de tiempo de un programa compilado respecto al mismo interpretado no depende de:

    a) El compilador.

    b) La máquina.

    c) El nivel del lenguaje.

    d) Del programa.

20. El hecho de que un programa nos presente unas facilidades que permitan ignorar las propiedades propias del microprocesador se denomina máquina

    a) Ficticia.

    b) Oculta.

    c) De alto nivel.

    d) Virtual.

18.3 GENERALIDADES DE LOS LENGUAJES DE PROGRAMACIÓN

En este apartado vamos a estudiar las características sintácticas de los lenguajes de programación, es decir, nos vamos a fijar en el cómo se escriben, y sobre todo en los elementos que integran un lenguaje.

En el primer apartado ya hemos hecho un resumen de lo que era el BASIC en conjunto. Ahora vamos a estudiar cómo se modifican estas características en otros lenguajes.

18.3.1 Los objetos de los lenguajes

Todos los lenguajes manipulan dos objetos: las *variables* y las *constantes.*

Tipos de objetos de un lenguaje

Las diferencias surgen en el *tipo* de variables y constantes que utiliza cada lenguaje.

El tipo de las variables se clasifica según el grado complejidad en tipos *primitivos* y tipos *compuestos.*

18.3.1.1 Tipos primitivos

Los tipos primitivos son los fundamentales del lenguaje. No es posible imaginar la manipulación de elementos más simples dentro del lenguaje.

En los lenguajes de programación los tipos primitivos son:

a) Tipo *char:* El nombre *«char»* proviene de la palabra inglesa «character» que significa carácter. Se refiere a un número entero de 0 a 255 o bien de −128 a 127 si se considera con signo. Ocupa generalmente un byte de memoria, es decir, 8 bits.

Seguramente le llama la atención que los valores de las variables de tipo *char* puedan ser interpretadas con signo o sin signo. En el capítulo 6 del segundo tomo, al hablar de los valores lógicos se estudió la manera de cómo se tratan las variables con signo o sin él. El último bit puede indicar el signo o no. En cualquier caso, a nivel interno las operaciones son idénticas; la diferencia aparece cuando interpretamos este número, es decir, cuando lo transformamos en una cadena de caracteres.

Char es el número que cabe en un byte

No confunda una variable de tipo *char* con una cadena de caracteres, o mejor una cadena de caracteres que contiene un carácter. Son entidades bien diferenciadas, ya que una cadena de caracteres es en último término una tabla de caracteres de longitud variable. Por lo tanto, puede tener cero caracteres, uno, dos, etc. En cambio una variable de tipo char es simplemente un número que cabe en un byte.

b) Tipo *integer:* La palabra *«integer»* proviene del inglés y en castellano significa entero; es decir, valores sin decimales. En términos generales, este tipo de variable se utiliza para contador. Los lenguajes de programación suelen tener varios tamaños de estos números, además de poderlos tratar con signo o sin signo. Tamaños típicos son contadores que van desde 0 hasta 65535 o de −32768 a 32767, que corresponden a 16 bits de memoria; contadores que van desde 0 hasta 4294967295 o de −2147483648 a 2147483647 que corresponden a 32 bits de memoria.

Diversas longitudes para los enteros

En la mayoría de los lenguajes de programación se permite escribir los números en base decimal y además en base 16 (hexadecimal), en base 8 (octal) y en base 2 (binario). Ciertamente no todos tienen todas las posibles representaciones pero casi siempre hay alguna de ellas como alternativa a la notación decimal.

c) Tipo *lógico:* Suelen ser como en el BASIC valores equivalentes a tipos enteros. De hecho almacenan valores que se asocian a un SÍ o un NO. Universalmente el valor cero se asocia al NO y generalmente el valor distinto de cero al valor SÍ. Observe que en muchos lenguajes de programación el valor del SÍ no está asociado a un único valor sino que simplemente se enuncia como distinto de cero.

d) Tipo *real:* Son los tipos de valores que contienen números con decimales. En la mayoría de lenguajes suelen haber números reales de varias precisiones; es decir, que permiten almacenar un mayor o menor número de decimales. Incluso en varios dialectos del BASIC existen números que se denominan de doble precisión. La manera de saber cuántos decimales puede contener se puede averiguar con el siguiente programa (se escribe en BASIC porque es el lenguaje que conocemos, pero su esquema es idéntico en los demás lenguajes)

```
10 LET N=1
20 IF 1. + N = 1. THEN GO TO 50
30    LET N=N/2
40    GO TO 20
50 PRINT N
```

Precisión de la máquina

Este programa empieza colocando la variable N al valor 1. En la línea 20 es la instrucción más difícil, se compara 1. + N con el 1, si son iguales se imprime N y si son distintos se divide N por 2 para ir otra vez a la pregunta.

La pregunta que se realiza en la línea 20 parece que no tenga sentido pues nunca puede ser igual. Esta afirmación es cierta si la máquina de calcular es de precisión infinita; es decir, con un número de decimales infinito. Si la precisión es finita, es decir, con un número de decimales máximo, como en el caso de un ordenador, siempre se encuentra un valor los suficientemente pequeño de N para que sumado a 1 dé como resultado 1, ya que no se soportan todos los decimales posibles.

Si quiere seguir el programa añada la instrucción

```
25 PRINT 1. + N
```

y verá como va evolucionando el número.

No confunda la precisión con el número más pequeño que puede utilizar la máquina. En general, todos los lenguajes de programación permiten manipular números mucho más pequeños que los de la precisión. La precisión nos indica solamente que al hacer operaciones de suma entre dos números que son muy distintos, sólo podemos almacenar un cierto número de decimales. Así la mayoría de máquinas permiten utilizar un número tan pequeño como 0.000000000000001 (o 1.E–15 en notación científica) en cambio la mayoría de ellas al sumar este número a 1 da como resultado 1 ya que no soportan 15 decimales.

En todos los lenguajes de programación se soporta la notación científica.

La manera de escribir en notación científica da idea de cómo se trata el número internamente. En general, se utilizan cuatro partes para expresar un número real; en primer lugar el signo del número, después un número comprendido entre 0 y 1, después un signo del exponente y

finalmente un número para indicar el exponente. La base numérica en que se expresa suele ser la 2, pero no nos debe preocupar demasiado, ya que en todo proceso de escritura se realizan las conversiones oportunas al sistema decimal que estamos acostumbrados a utilizar.

e) Tipo *puntero:* Este tipo de objeto primitivo sólo aparece en lenguajes que están cercanos al lenguaje ensamblador. Contiene como dato una dirección de memoria, que nos indica dónde está en la memoria del ordenador un objeto determinado.

Esencialmente se trata de un número entero sin signo, ya que las direcciones de memoria son de este tipo. Existe, sin embargo, una diferencia sutil entre puntero y dirección de memoria. En un tipo puntero, aparte de la dirección de memoria, se sabe a qué tipo de dato apunta. La dirección de memoria es simplemente un número sin indicación alguna de cuál es el dato contenido en ella; por lo tanto, no sabemos cómo hay que interpretar este contenido, ¿cómo un entero?, ¿cómo un real? En cambio, en un puntero se sabe cómo debe ser interpretado el contenido.

No todos los lenguajes disponen de todos los tipos primitivos. Es más, existen algunos lenguajes que los tipos primitivos son mucho más elaborados. De todas maneras como norma general los tipos mencionados son los más frecuentes.

18.3.1.2 Tipos compuesto o estructurados

Los tipos compuestos son los que se construyen a partir de los primitivos mediante agrupaciones.

18.3.1.2.1 Tablas

La agrupación prácticamente universal en los lenguajes de programación es la *tabla.* La manipulación de estas estructuras se hace mediante los subíndices y una declaración de la reserva de memoria que debe hacerse para operar con la agrupación.

18.3.1.2.2 Registros o estructuras

Otro tipo de agrupación es el *registro* o *estructura.* La idea básica es agrupar cosas heterogéneas; es decir, distintas, en una misma entidad. En este sentido, es un concepto contrapuesto a las tablas que, por definición, son cosas exactamente iguales, o en términos más técnicos, homogéneas.

Este tipo de agrupación es parecida a los registros de los ficheros pero utilizados en la memoria del ordenador. Se utilizan para definir propiedades de un objeto compuesto. Por ejemplo, una persona se caracteriza por su nombre, primer apellido, segundo apellido, año de nacimiento y peso. En el lenguaje de programación C esto se expresa de la manera siguiente:

```
typedef struct
        {
            char  nom[15]; /* Nombre */
            char  pap[20]; /* Primer apellido */
            char  sap[20]; /* Segundo apellido */
            int   año    ; /* Año de nacimiento */
            int   peso   ; /* Peso */
            }
  PERSONA;
```

Como puede observar en estas agrupaciones toman objetos primitivos distintos, tales como el char y el int (significa integer). Y no tan sólo tipos primitivos, sino que utiliza tablas de caracteres como componentes dentro de esta estructura de registro. Los corchetes detrás de los nombres indican el tamaño máximo de caracteres que puede tener el nombre o los apellidos.

Los nombres que aparecen en la estructura se denominan los *miembros del registro.* En este ejemplo estos miembros son nom, pap, sap, año y peso.

La ventaja de este tipo de agrupación es que se puede manipular como una entidad entera y sin estructura, la entidad PERSONA en nuestro ejemplo, o bien se puede acceder a cada uno de los miembros para cambiar u operar con el valor.

Por ejemplo, si en el registro anterior en lugar de guardar el año de nacimiento deseamos guardar el día completo, es posible expresar esto mediante la adición de la estructura fecha que es:

```
typedef struct
        {
           int día; /* Día */
           int mes; /* Mes */
           int año; /* Año */
           }
  FECHA;
```

Un registro puede contener una referencia a otro registro

Entonces el registro PERSONA se expresa como

```
typedef struct
        {
            char  nom[15]; /* Nombre */
            char  pap[20]; /* Primer apellido */
            char  sap[20]; /* Segundo apellido */
            FECHA nac    ; /* Día de nacimiento */
            int   peso   ; /* Peso */
            }
  PERSONA;
```

Observe que la fecha puede tratarse como una tabla de tres enteros. Se prefiere esta forma porque aunque son números, sus significados son muy diferentes.

Los miembros de un registro pueden ser a su vez otros registros. Esto lleva a poder utilizar datos que se denominan estructurados, ya que con la utilización de un registro es posible dotar de una estructura en muchos niveles de los datos que manipulamos.

18.3.1.2.3 Cadenas de caracteres

Las *cadenas de caracteres* son también una agrupación que poseen muchos de los lenguajes de programación. Aunque una cadena de caracteres es una tabla de caracteres, se considera distinta de la tabla de caracteres porque el lenguaje permite manipularlas sin tener en cuenta su longitud. La administración de memoria se hace de manera automática y de manera invisible para el usuario. Esta es la manera cómo el BASIC trata los tipos textuales.

18.3.2 Las expresiones

En todos los lenguajes de programación se pueden formar expresiones, mediante los objetos del lenguaje y los operadores. De hecho, los operadores que posee el BASIC se encuentran en todos los lenguajes de programación. En este sentido el BASIC es muy completo, o al menos tan completo como muchos de los demás lenguajes de programación.

El tipo de operadores y la manipulación de las precedencias de las expresiones es una característica estándar de los lenguajes. Alguno de ellos amplía notablemente las operaciones básicas, pero se trata más de una excepción que una regla. El lenguaje C que se ha mencionado es el que aporta una variedad más grande de operadores, pero mantiene los que tiene el BASIC con las mismas precedencias. Los nuevos operadores que introduce son para tratar objetos primitivos muy especiales, que son los punteros.

18.3.3 La asignación

En todos los lenguajes de programación existe el concepto de *asignación* como medio para cambiar los valores de las variables.

La forma de expresarla más extendida es con el signo igual. Pero existen lenguajes que utilizan símbolos como los dos puntos y el igual (:=) o una flecha hacia la izquierda (〈−).

Cuando existen datos estructurados la asignación puede significar un movimiento notable de memoria ya que cuando se manipulan registros la estructuración pueden llevarnos a entidades grandes.

18.3.4 Las instrucciones de entrada y salida

Es el aspecto menos estándar en todos los lenguajes. Cada uno tiene una filosofía particular para tratarlas. En general, suele ser donde se debe dedicar más esfuerzo al aprender un nuevo lenguaje de programación. Es especialmente complicado en el tratamiento de ficheros de disco, tópico que no corresponde exactamente a un lenguaje de programación, pero que ciertamente a nivel profesional está muy relacionado.

18.3.5 Las instrucciones de control

Según las instrucciones de control un lenguaje se denomina estructurado o no estructurado. Cuanto más moderno es un lenguaje incorpora **más elementos estructurales de control de flujo en el sentido que se ha descrito en el capítulo 13 del tomo anterior.**

En primer lugar, todos los lenguajes de programación contienen las instrucciones elementales de control que son: el salto incondicional (GO TO), el salto condicional (IF ... THEN) y el salto a subrutina (GO SUB).

Ud. ya se ha dado cuenta cuando se escribe un programa y hay que utilizar el GO TO son muy engorrosas las referencias hacia adelante; es decir, hay que enviar el flujo del programa a un lugar que aún no hemos escrito.

El concepto de etiqueta

En general, en este tipo de lenguajes las instrucciones no van numeradas como en el BASIC. Se asume que una instrucción va detrás de la otra en el orden en que están escritas. Cuando hay que utilizar un GO TO se introduce el concepto de *etiqueta* que es un tipo especial de nombre que identifica un punto del programa para que la referencia con el GO TO sea inequívoca. Este elemento suele aliviar notablemente este problema

En los lenguajes de tipo estructurado se intenta evitar esta referencia hacia adelante mediante la utilización exhaustiva de elementos de abrir y cerrar, como los paréntesis. En BASIC el ejemplo claro es la estructura FOR/NEXT. Iniciamos el bucle con un FOR y acabamos con un NEXT en el momento que nos convenga.

Concepto de bloque secuencial

El concepto de bloque secuencial está desarrollado en estos lenguajes estructurados, de tal manera que una instrucción se puede componer de varias instrucciones más elementales.

Por ejemplo, en PASCAL y en ALGOL se utiliza la palabra *begin* para iniciar un bloque y la palabra *end* para finalizarlo; en el lenguaje C se utiliza la llave { para iniciar un bloque y la llave } para cerrar el bloque. Todas las instrucciones en el interior de estos elementos se tratan como una única instrucción, en el sentido más estricto de bloque secuencial tal como se ha definido en la lección 13.

En este tipo de lenguajes las instrucciones de control suelen tener la forma de escribirlos con una única instrucción. Por ejemplo, en PASCAL un caso de alternativa simple es

```
if ( ⟨condición⟩ ) then
                    ⟨bloque secuencial⟩
else
                    ⟨bloque secuencial⟩
```

Al tener bien definido el bloque secuencial las instrucciones de control se escriben de una manera muy sencilla. Otros ejemplos puede ser:

while (condición) ⟨bloque secuencial⟩

while es la palabra inglesa que es equivalente a mientras y se trata de una estructura mientras.

do ⟨bloque secuencial⟩ while (condición)

es la forma para una estructura iterativa hasta *do* es la palabra inglesa, que en castellano significa hacer. En este ejemplo concreto, en lugar de utilizar hasta que se dé una condición, se utiliza mientras se cumpla la condición.

En los lenguajes que no disponen de la estructura de bloque de instrucciones, las instrucciones de control estructuradas aparecen de una manera mixta (en la actualidad hay muchos dialectos del BASIC que incorporan este tipo de instrucciones).

Las instrucciones con terminación para estructurar el programa

Por ejemplo, el caso de alternativa simple en un BASIC estructurado se escribe como:

```
IF ⟨condición⟩
   THEN
        ⟨instrucción⟩
         ....
        ⟨instrucción⟩
   ELSE
        ⟨instrucción⟩
         ....
        ⟨instrucción⟩
   IFEND
```

Todas las instrucciones van numeradas aunque no se coloque la numeración en el esquema anterior. La palabra IFEND marca el fin de la estructura. De hecho este es el fin implícito de la estructura. Cuando se ejecuta la parte THEN el programa va a buscar el IFEND más próximo. De la misma manera, cuando la condición es falsa el programa va a buscar la palabra ELSE más próxima.

Así también un bucle MIENTRAS, en BASIC estructurado se escribe como

```
WHILE ( ⟨condición⟩ )
        ⟨instrucción⟩
        ....
        ⟨instrucción⟩
        WEND
```

Las instrucciones van numeradas como en el caso anterior. La palabra WEND nos marca el final del bucle de la misma manera que en el caso del FOR y el NEXT.

Variables globales y locales

En cuanto a las llamadas a las subrutinas, en todos los lenguajes compilados se utilizan argumentos como en las funciones del BASIC. Además, las variables que se definen dentro de una subrutina no tienen nada que ver con las variables definidas en el programa principal. Esto es lo que se denominan *variables locales.* En BASIC este concepto no existe, ya que todas las variables se pueden utilizar con el mismo valor desde cualquier parte del programa. En este caso se denominan *variables globales.*

En muchos de los lenguajes compilados las subrutinas se parecen más a las funciones del BASIC, incluso en el sentido de operadores, que a las subrutinas del BASIC.

18.4 ALGUNOS LENGUAJES DE PROGRAMACIÓN

En este apartado vamos a repasar las características de los principales lenguajes de programación. Necesariamente es una visión muy breve pero intentaremos mostrar las características más sobresalientes de cada uno de ellos.

La selección de los lenguajes se ha hecho con un criterio personal, pero creemos que cubre más de un 80 % de la programación profesional en lenguajes de interés general. Tenga en cuenta que existen gran cantidad de lenguajes que se diseñan pero que en realidad no cuajan por diversas razones, a veces simplemente comerciales.

18.4.1 FORTRAN

El nombre proviene de la abreviatura de la frase inglesa «Formula Translator» que significa en castellano Traductor de fórmulas. El lenguaje es muy antiguo (para la informática). En noviembre de 1954 la Compañía IBM publica la descripción preliminar del lenguaje. Durante estos 32 años el FORTRAN ha evolucionado y se han dado diversas revisiones. La que actualmente se considera como la estándar es el compilador llamado FORTRAN 77.

Este lenguaje está orientado a resolver problemas científicos tal y como indica su nombre. Ciertamente, después del BASIC es el lenguaje más popular, aunque en estos momentos su interés decrece ya que ha dejado de ser el lenguaje adoptado para el desarrollo de software del departamento de defensa de los Estados Unidos, el mayor consumidor de programas del mundo.

Es en lenguaje que sólo se utiliza compilado. En líneas generales es muy parecido al BASIC, en cuanto a operadores y objetos primitivos. Incluye, no obstante, los tipos lógicos y no tiene las cadenas de caracteres. Dispone de tablas de caracteres que se tratan como conjuntos dimensionados, pero permite operaciones tales como la concatenación de textos, aunque dejando estos textos a una longitud fija siempre.

Posee los números enteros de varias longitudes y números reales de varias precisiones.

Las instrucciones de control en las últimas revisiones incorporan los elementos de la programación estructurada, pero mantienen los esque-

mas iniciales para hacer los programas antiguos compatibles con el compilador.

El FORTRAN sólo tiene la tabla como dato estructurado

No incorpora ningún tipo de dato estructurado a excepción de las tablas.

Las entradas y salidas del programa se fundamentan en las instrucciones READ y WRITE, equivalentes a las INPUT y PRINT del BASIC pero algo más engorrosas de manipular. Llevan asociado una instrucción FORMAT que es como una plantilla de cómo hay que leer los datos. En las versiones más antiguas y más modernas hay una instrucción PRINT que permite manipular la salida con más comodidad.

Utiliza las subrutinas y las funciones para descomponer el programa en módulos. El concepto de variables locales se aplica en la construcción de las subrutinas y funciones. Por lo tanto, se basan los argumentos para ejecutar en una subrutina y en una función.

La subrutina en FORTRAN se diferencia de la función porque en la primera es posible acceder a algunos datos del programa principal, mientras en la segunda esto está totalmente prohibido.

18.4.2 COBOL

El nombre es la abreviatura de la frase inglesa Common Bussines Oriented Language, que en castellano se traduce como Lenguaje orientado a los negocios. Prácticamente es de la misma época que el FORTRAN. Se lanza al mercado en 1956 y está diseñado para resolver programas en el ámbito de la gestión de empresas. También ha evolucionado mucho desde los inicios pero su estructuración ha sido menor que el FORTRAN.

Lenguaje de orientación a la gestión

Aunque inicialmente se orientó a negocios puede considerarse un lenguaje de interés general.

Se trata de un lenguaje compilado. Se organiza en diversas partes que se llaman divisiones y en cada una de ellas se organizan las comunicaciones, el cálculo, etc. La evolución del COBOL inicialmente fue facilitar los accesos a los ficheros de datos, indispensables en los problemas de negocios. Actualmente esta tarea la absorbe el llamado sistema operativo que, como hemos visto en el capítulo 17, sirve para facilitar al usuario un acceso más ordenado y estandarizado a los recuerdos del ordenador.

Desde el punto de vista de los objetos que trata manipula números y tablas de caracteres. Incorpora también el concepto de registro en sus datos básicos y la tabla.

Los operadores no son símbolos sino palabras que en inglés son equivalentes a los símbolos, así para sumar se utiliza la palabra ADD que significa justamente añadir, sumar.

18.4.3 PL/I.

Es un lenguaje promovido por la casa IBM que pretende ser una síntesis de los dos lenguajes anteriores. Es decir, conseguir un lenguaje que permita con igual comodidad tratar problemas científicos y problemas comerciales.

En sus elementos básicos hay los primeros elementos de programación estructurada.

Introduce en su juego básico de tipos la cadena de caracteres.

18.4.4 ALGOL

En 1960 se inicia un movimiento para realizar un lenguaje de programación que incorpora todos los avances teóricos producidos en la teoría de la programación. En 1966 aparece la primera definición formal del lenguaje. Se le da el nombre correspondiente a la abreviatura de «Algorithmic Language», que significa lenguaje algorítmico. Incorpora como novedad principal el concepto de bloque secuencial de instrucciones y se apuntan las primeras construcciones sintácticas estructuradas. En 1968 se da una segunda definición del lenguaje que incorpora novedades como la manipulación de tablas que pueden hacerse mayores o menores a lo largo del programa. Los datos estructurados están también en el lenguaje y una novedad que parecía definitiva, la posibilidad de definir operadores por el usuario.

Este lenguaje no prosperó como herramienta profesional quizá por el hecho de ser un proyecto demasiado ambicioso, que se perdió en los aspectos sintácticos olvidando la semántica, o bien porque no tenía un respaldo de ninguna casa comercial, y por lo tanto, no había interés en invertir en este proyecto tan costoso, o por ambos a la vez.

Lenguaje de los teóricos

Sin embargo, es un lenguaje utilizado por los teóricos de la programación como vehículo de expresión en las publicaciones.

El lenguaje es compilado. En líneas generales es un lenguaje que es difícil de aprender y de manipular, si no se conocen otros lenguajes.

18.4.5 PASCAL

Después del fracaso del ALGOL se inicia por parte de algunos de los miembros de la comisión de redacción de las reglas del lenguaje ALGOL la revolución de la programación estructurada cuyo nombre más significativo es Dijkstra. El profesor suizo Niklaus Wirth crea, bajo la luz de la programación estructurada, la definición de un lenguaje que quiere utilizar para la enseñanza de programación. Era el año 1975. A este lenguaje le denomina PASCAL en memoria a un famoso matemático y físico francés del siglo XVII.

La novedad más importante que aporta este lenguaje es que responde a un diseño previo estructurado, y con respeto a la tradición incorpora el GO TO aún cuando no es necesario para la realización de los programas, las herramientas de programación estructurada son suficientes para que esta instrucción pueda ignorarse.

Introduce las estructuras de datos

Adopta también la estructuración de datos como herramienta básica para el desarrollo de programas.

En este aspecto, la novedad que presenta es la introducción en el lenguaje de los objetos de tipo puntero y la definición de los operadores (por consiguiente de las operaciones) para manejarlos.

Introduce en su lenguaje gran cantidad de elementos matemáticos, o mejor, construcciones sintácticas para tratar los elementos que son comunes a las Matemáticas tales como los conjuntos.

Por lo demás las expresiones son parecidas al resto de lenguajes de orientación científica como puede ser el BASIC o el FORTRAN.

El resultado es un compilador no demasiado grande que permite incorporar este lenguaje en la mayoría de microordenadores y ordenadores personales, esta particularidad es la que ha dado gran popularidad al PASCAL y en la actualidad es uno de los lenguajes más utilizados, al menos a nivel de programadores para máquinas relativamente pequeñas.

18.4.6 C

El C es un lenguaje desarrollado por Ritchie y Kernighan en los Laboratorios Bell (compañía telefónica de los Estados Unidos) desde 1970 hasta 1978 en que se publica el texto definitivo de la definición del lenguaje.

El C es un lenguaje de programación general, es decir, sirve, para desarrollar cualquier tipo de problema. Esta precisión es importante, porque se asocia al C al desarrollo de sistemas operativos, es decir, programas especializados en ciertas tareas.

En un lenguaje de bajo nivel, es decir, cercano al assembler, en este sentido la gran virtud del lenguaje C es que permite graduar mucho el nivel del lenguaje. Un mismo programa se puede escribir con un parecido sorprendente al FORTRAN o bien con construcciones más típicas del assembler. Naturalmente el assembler suele ser más eficiente que el FORTRAN, la virtud del C es que se puede poner más énfasis en la eficiencia en los puntos en que el programa lo necesita, en cambio se puede programar con mucha comodidad en las zonas en donde la eficiencia no es tan importante.

El control de flujo es totalmente estructurado y también permite la estructuración de los datos en el sentido que se ha mencionado en el PASCAL.

No existen subrutinas pero si funciones. Para ser exactos, las funciones son el núcleo de un programa desarrollado en C. Estas funciones son como las subrutinas en el sentido de que se pasan argumentos y pueden acceder a los datos del programa principal, pero son funciones porque en cualquier circunstancia devuelven un valor.

No posee ninguna instrucción de entradas y salidas

Curiosamente este lenguaje no tiene ninguna instrucción de entrada y salida todas se realizan mediante las llamadas a unas funciones de una biblioteca que se suele suministrar con el compilador. Este esquema permite gran flexibilidad en la entrada y la salida que, como ya hemos visto, es de las instrucciones más difíciles de estandarizar. Una manera de eliminar problemas es ignorando estas instrucciones como tales instrucciones en el lenguaje.

El C ha tenido un auge enorme en los últimos años, precisamente porque permite cambiar de máquina sin gran esfuerzo de programación y además, como el compilador es pequeño, se puede utilizar con comodidad en la mayoría de ordenadores personales.

El lenguaje es compilado, aunque se ha hecho algún intento para conseguir un interpretador. De todas maneras hay que considerarlo como un lenguaje que se compila.

18.4.7 Lenguajes interpretados

A continuación se describen lenguajes interpretados como el BASIC. Se pueden considerar de interés general pero son bastante distintos al resto de los lenguajes, incluido el propio BASIC.

18.4.7.1 APL

Es un lenguaje especializado en el tratamiento matemático de matrices y vectores. Su característica más curiosa es que utiliza un juego de símbolos especiales muy superior al resto de los lenguajes de programación. Esto obliga a poseer un teclado especial para realizar programas en él.

18.4.7.2 LISP

Lenguaje de la inteligencia artificial

Las siglas corresponden a «List Processing» que en castellano significa procesador de listas. Este es el lenguaje más antiguo. Curiosamente es anterior al FORTRAN y al COBOL. Actualmente ha resucitado del olvido debido a que es necesario para el desarrollo de los algoritmos de inteligencia artificial.

Se llama inteligencia artificial a una especialidad de la informática que trata de simular los procesos de razonamiento o reconocimiento de formas humano. Un ejemplo de un programa de inteligencia artificial es el que es capaz de reconocer la lectura de un manuscrito. El saber leer la letra manuscrita de otra persona no es una tarea fácil, a veces ni por las propias personas.

Este lenguaje se diferencia de los demás en que el objeto fundamental y único del lenguaje son las listas. Las expresiones matemáticas son listas de símbolos que se manipulan según las reglas de la lógica.

Toda la potencia del lenguaje es la manipulación simbólica, antes que el cálculo numérico.

18.4.7.3 LOGO

Es un lenguaje muy parecido en sus aspectos fundamentales al anterior pero está especializado en los procesos de aprendizaje.

Con el fin de evitar el nivel de abstracción en que se mueve el lenguaje anterior, se dota al lenguaje de unos elementos primitivos que son las instrucciones necesarias para realizar dibujos.

La característica esencial es que permite enseñar al ordenador nuevos conceptos creados por uno mismo para hacer construcciones cada vez más elaboradas. Por ejemplo, se crea el concepto cuadrado a partir de cuatro líneas. Este concepto puede utilizarse, a partir de este momento, incorporado al lenguaje.

Su utilidad se centra en la educación, ya que permite al niño organizar de una forma lógica y coherente su razonamiento. Aunque no es propiamente un lenguaje de programación es interesante por presentar unas características educacionales que sirven también para iniciarse en la programación.

18.5 LOS PROGRAMAS DE APLICACIÓN

La importancia que han adquirido recientemente los ordenadores personales ha provocado que el número de aplicaciones, es decir, programas para sacar provecho de estos ordenadores, se haya multiplicado considerablemente.

Las listas de programas son muy largas. Sin embargo, y ya con la perspectiva de varios años, es necesario hacer referencia a algunas de ellas porque se han mostrado una herramienta muy útil en el trabajo individual.

Vamos a discutir áreas de aplicación más que programas o lenguajes muy específicos concretos. En esta discusión vamos a intentar resaltar los aspectos de utilidad y uso antes que las instrucciones específicas. Estamos convencidos de que el futuro de la informática pasa por la utilización de estas herramientas, ya sea para continuar el desarrollo informático, ya sea para simplemente aumenta el rendimiento de nuestro trabajo personal.

Las aplicaciones básicas que deben considerarse en un orden personal son tres:

a) Procesadores de texto.

b) Hoja electrónica.

c) Bases de datos.

18.5.1 Procesadores de textos

Se entiende como procesador de texto un programa que ayuda en la elaboración de textos escritos.

Esta Enciclopedia se ha escrito utilizando un procesador de textos.

¿Que ventajas aporta un procesador de textos sobre la escritura normal?

Ventajas de los procesadores de texto frente a las máquinas de escribir

Estas ventajas son:

a) El proceso de escribir como actividad de teclear se desliga de la actividad de imprimir sobre papel.

b) Se efectúan tantas correcciones como se desean sin necesidad de tener que repetir lo que está impreso y no hace falta modificarlo.

c) Se memoriza el texto sobre un soporte magnético.

d) La presentación del texto puede variarse sin necesidad de volver a teclear el texto.

En algunos procesadores se pueden obtener ventajas adicionales, tales como inclusión de gráficos, elaboración de fórmulas, corrección de palabras, mezcla de varios textos, etc.

La filosofía fundamental es que la pantalla del ordenador sirve de papel para el momento de la entrada del texto. Un programa que permite movernos arriba y abajo de la pantalla, hacia la derecha y a la izquierda es el núcleo fundamental de esta parte.

A medida que se entra se puede ir memorizando a disco. Es conveniente hacerlo, pues si se corta la electricidad lo que tenemos en el ordenador desaparece a menos de que se haya grabado en una memoria permanente como es el disco.

Cuando estamos satisfechos de nuestro trabajo procedemos a imprimir el resultado. Después de repasar el escrito podemos volver a retocar y volver a imprimir las partes que deseamos.

En el mercado hay muchos programas de procesamiento de textos y la selección es una cuestión delicada. La primera reflexión que debe hacerse es que cada uno de ellos tiene una filosofía distinta y debe escoger la que mejor se adapte a su manera de trabajar, aunque tenga menos operaciones que otro. El 90 % del tiempo que usamos un programa de este tipo tecleamos y no utilizamos operaciones especiales. No existe un procesador de textos perfecto.

En segundo lugar, considerar qué tipo de trabajo va a realizar, ya que esto le debe dirigir hacia los procesadores de texto que tengan las operaciones que facilitan al máximo el trabajo. A veces es conveniente hacer un tipo de trabajos con un procesador de textos y otro tipo con otro procesador. Desde el punto de vista económico son programas accesibles.

Operaciones básicas de un procesador de texto

Las operaciones básicas que suele llevar un procesador de textos son:

a) Movimientos del cursor. En esta clase es necesario que existan las cuatro funciones de movimiento de la cabeza de escritura, arriba, abajo, derecha e izquierda. Además es conveniente que se disponga de los movimientos de página arriba y abajo. Situación en el interior de una línea, al principio y al final. Al principio y al final del documento. Menos importante, pero interesante, es la situación en palabras sucesivas tanto hacia adelante como hacia atrás.

b) Modo de inserción y de sustitución. Estos dos modos permiten definir qué acción toma el procesador de textos cuando se intenta escribir sobre una letra. En el modo de inserción no borra la letra sino que la nueva letra tecleada se sitúa en el sitio de la antigua y se corre la antigua hacia la derecha. En el modo de sustitución simplemente la letra nueva sustituye a la antigua sin provocar ningún movimiento del texto.

c) Operaciones de recortado y pegado. Se trata de operaciones que permiten seleccionar un trozo de texto, tal como un párrafo, para cambiarlo de lugar, borrarlo entero, o repetirlo en otro lugar.

d) Selección de tabuladores. Permiten definir, como en la máquina de escribir, puntos de detención en una línea, que sirven para realizar columnas.

Figura 8. Operaciones más frecuentes en un procesador de textos.

```
OPERACIONES DEL PROCESADO DE TEXTOS

Movimientos cursor: No alteran el texto.

     Cursor Arriba.          Palabra anterior.       Página Arriba.
     Cursor Abajo.           Palabra siguiente.      Página Abajo.
     Cursor Izquierda.       Inicio de línea.        Inicio Documento.
     Cursor Derecha.         Fin de línea.           Final Documento.

     Tabulador.              Fijar tabuladores.

     Buscar una frase o palabra.

Movimientos de texto: Alteran el contenido del texto.

     Tecla de borrado y retroceso.       Inserción de caracteres.
     Tecla de borrado.
     Borrar palabra.                     Sustituir palabra.

     Copiar un bloque de texto.          Copiar de otro archivo.
     Mover un bloque de texto.
     Borrar un bloque de texto.

     Ajustar un párrafo.
     Centrar una línea.

Opciones de impresión.

     Fijar el margen derecho.            Cabecera de página.
     Fijar el margen izquierdo.          Pié de página.
     Longitud de la página.

     Sangrado de inicio párrafos
```

e) Operaciones de formateo. Tales como centrar un título, ajustar a la derecha un párrafo, etc. Este mecanismo de alineación se hace añadiendo blancos al azar repartido en el interior de la línea de tal manera que quede la última letra en el margen derecho.

En la figura 8 se da un conjunto completo de operaciones de un procesador de textos.

18.5.2 La hoja electrónica

Se entiende como hoja electrónica una máquina de calcular que trabaja sobre un soporte de filas y columnas; es decir, como en un papel.

Esta máquina de calcular se diferencia de las tradicionales en que allí sólo se dispone de la visualización de un número y en este tipo de programa se dispone de un visor al estilo del presentado en la práctica

Figura 9. Cálculo de costes con la hoja electrónica.

	1	2	3	4	5
1	Coste 1Kg. Embutido A-23				
2					
3	Costes Directos	Cant.	Precio	Total	%
4					
5	Carne de Cerdo de 1a.	0,40	300	120	35,82
6	Carne de cerdo de 2a.	0,30	210	63	18,80
7	Canal de Cerdo	0,15	100	15	4,48
8	Grasa de cerdo	0,15	80	12	3,58
9		--------			
10		1,00			
11	Merma cocción 3%	0,03			
12		--------			
13	Total de Kgs.	0,97			
14					
15	Bolsa de plástico	1	25	25	7,46
16	Especies	1	40	40	11,94
17				--------	
18				275	82,08
19	Costes Indirectos				
20					
21	Transportes		23		
22	Mano de obra		15		
23	Gatos generales		10		
24	Impuestos		2		
25			--------		
26			50	50	14,92
27				--------	
28				325	
29	Precio unitario			335	

del capítulo 12, en el tomo tercero. Por otra parte dispone de una cantidad de memoria bastante más grande que una máquina de calcular normal.

La figura 9 muestra una presentación típica de la utilización de la hoja electrónica. Por ejemplo, en el cálculo de coste, se presente el caso del cálculo del coste de 1 Kg. de un embutido.

Este ejemplo consta de 5 columnas que van rotuladas con Costes Directos, Cantidad, Precio, Total y Tanto por ciento.

La primera columna es de tipo alfanumérico que permite describir los materiales que intervienen en la fabricación del embutido. La segunda columna es numérica e indica qué cantidad de cada elemento contiene. La tercera, el precio de cada uno de estos elementos. Estos serían los datos que se deben llenar.

La columna Total contiene los cálculos de multiplicar la columna de cantidad con la de precio. Finalmente la columna del Tanto por ciento nos indica cual es el valor de cada producto en relación al precio final.

A la hora de programar esta hoja electrónica en las columnas del total está escondida una fórmula, que es realmente lo que hemos entrado, que permite hacer el cálculo. Por ejemplo la fórmula que debemos entrar en la fila de Carne de Cerdo de primera y la columna del Tanto por ciento es:

RC[−1]/R29C4∗100

que indica que se toma el valor correspondiente a la misma fila y columna anterior (RC[−1]) se divide por el precio total que está en la columna 4 y fila 29 (R29C4) y se multiplica este valor por cien.

La gran ventaja de este sistema es que si ahora nos varia un precio, la simple sustitución del precio viejo por el nuevo en la casilla correspondiente hace que inmediatamente todos los cálculos quedan rehechos.

18.5.3 Las bases de datos

La aplicación de bases de datos permiten mantener un fichero, en el sentido de fichero manual, que contienen datos.

La aplicación se basa en un primer elemento que consiste en la definición de la ficha. Consiste en un programa que nos permite definir los campos y los títulos que va a tener la ficha que deseamos. Este proceso es semejante a pedir a una imprenta que nos haga un modelo de ficha determinado.

Una vez hemos diseñado una ficha se procede a rellenarla para cada uno de las entidades de las que deseamos tener ficha. El ejemplo más inmediato es un fichero de cliente, en los que consta el nombre, la dirección, las condiciones de pago.

La tarea necesaria pero más pesada es rellenar cada una de las fichas.

Una vez está el fichero lleno, hay otro módulo del programa que permite hacer listados ya sea de la ficha completa, de diversas partes de la ficha, por ejemplo, imprimir etiquetas para hacer envíos por correo, hacer una lista de las condiciones de pago, etc.

La selección de fichas

Además, en la confección de esta lista se pueden seleccionar sólo algunas fichas del fichero. Esta selección depende en parte de cómo hemos diseñado la ficha. Sólo se pueden seleccionar por propiedades entradas y del grado de complejidad que posea la función de selección.

Como en el caso de los procesadores de texto, no porque el mecanismo de selección sea más complicado el programa es eficiente.

En general, son mejores mecanismos de selección sencillos pero de uso frecuente. Vale más tirar cuatro etiquetas que tener que montar un mecanismo muy complicado de selección.

Existen programas muy complejos que permiten manipular más de dos ficheros. Su uso a nivel personal no es recomendable por el esfuerzo que hay que hacer en aprender a diseñar correctamente las conexiones lógicas. Se trata de toda una especialidad de la informática.

18.6 Conclusión

Hemos llegado al final de la Enciclopedia, le hemos explicado muchas cosas que esperamos que sean útiles. Sin embargo, deseamos hacerle una última recomendación, la parte mas difícil de la informatica es tener claro el problema que se desea resolver.

Haga siempre el esfuerzo de pensar en la solución antes de sentarse delante de un teclado para hacer un programa. Los programas son buenos cuando resuelven correctamente un problema, no son buenos cuando no cumplan esta condición aunque haya utilizado para hacerlo las mejores técnicas de diseño que se conozcan.

RESUMEN

Todos los lenguajes manipulan dos objetos: las variables y las constantes.

Los tipos de variables pueden ser primitivos o compuestos.

Los tipos primitivos son:

a) Carácter (Char): Número que cabe en un byte.

b) Entero (Integer): Número entero sin decimales.

c) Lógico: Valor de cero y distinto de cero.

d) Real: Existen varias precisiones.

e) Puntero: Dirección de memoria que tiene un contenido conocido.

Los tipos compuestos o estructurados son:

a) Las tablas. Repetición de elementos iguales. Contenido homogéneo.

b) Registros o estructuras. Agrupa elementos heterogéneos, es decir, distintos.

c) Cadenas de caracteres.

Las expresiones son construcciones que se realizan con los objetos y los operadores. Existen en todos los lenguajes de programación.

La asignación se representa casi siempre por el símbolo igual, aunque hay lenguajes que utilizan otro símbolo.

Las instrucciones de entrada y salida son muy diferentes en cada lenguaje.

Las instrucciones de control dividen a los lenguajes en estructurado y no estructurados.

En general los lenguajes no estructurados no llevan las instrucciones numeradas y para utilizar el GO TO es necesario introducir el concepto de etiqueta.

Dentro de los lenguajes estructurados los hay que permiten el bloque secuencial de instrucciones.

En otros lenguajes, las estructuras incorporan el fin de estructura tal como el IFEND.

En la mayoría de lenguajes, las llamadas a subrutinas utilizan unas variables locales a la subrutina.

Los lenguajes de programación más conocidos y utilizados a criterio de los autores son: FORTRAN, COBOL, PL/I, ALGOL, PASCAL y C dentro de los compilados.

Los lenguajes interpretados son más de aplicación específica. Se cuentan en ellos además del BASIC, el APL, el LISP y el LOGO.

Tres tipos de programas de aplicación son los que se encuentran en los ordenadores personales:

a) Los procesadores de texto. Son programas que ayudan a la elaboración de textos desligando el teclado de la información de la escritura de la misma.

b) La hoja electrónica. Máquina de calcular simulada sobre una estructura de casillas.

c) Las bases de datos. Programas que sirven para simular los ficheros manuales con la ventaja de permitir una selección de la información mucho más potente que la manual.

EJERCICIOS DE AUTOCOMPROBACIÓN

Complete las frases siguientes:

21. Los tipos de objetos de un lenguaje son .... y compuestos.

22. Un tipo .... es una dirección de memoria con una indicación de cómo debe interpretarse el contenido.

23. La precisión de la máquina es el mayor número que sumado a uno da ....

24. Un .... es una agrupación de datos de diversos tipos.

25. En la definición de un registro puede aparecer una referencia a otro ....

26. Las instrucciones de entrada y salida son las .... estándar de un lenguaje.

27. Las instrucciones de .... son las que permiten dividir los lenguajes en estructurados y no estructurados.

28. Las tres aplicaciones más comunes en los ordenadores personales son el procesador de texto, ..., y las bases de datos.

29. El procesador de textos .... el proceso de tecleado del texto de la escritura sobre papel.

30. La hoja electrónica es como una máquina de ...., pero con mucha más memoria y dispuesta en forma de hoja de papel.

Rodee con un círculo la letra que corresponde a la respuesta correcta.

31. ¿Cuál de los siguientes tipos de objetos es compuesto?

 a) Entero.
 b) Cadena de caracteres.
 c) Real.
 d) Puntero.

32. Los lenguajes que no numeran las instrucciones utilizan para el envío mediante el GO TO.

 a) La etiqueta.
 b) Las tablas.
 c) Los bloques secuenciales.
 d) Los registros.

33. Cuando se utilizan subrutinas en los lenguajes compilados las variables que se usan son:

 a) Cadenas de caracteres.
 b) Argumentos.
 c) Globales.
 d) Locales.

34. Los lenguajes que tienen instrucciones de control más sencillas y más completas son los que utilizan:

 a) Las etiquetas.
 b) Los bloques secuenciales.
 c) Los finalizadores de estructura, tales como el IFEND.
 d) El GO TO estructurado.

35. El lenguaje de programación más antiguo es:

    a) FORTRAN.
    b) COBOL.
    c) C.
    d) LISP.

36. Los objetos de tipo puntero los introduce por primera vez el lenguaje.

    a) PASCAL.
    b) C.
    c) APL.
    d) COBOL.

37. El lenguaje que utiliza las listas como objeto básico y único del lenguaje es:

    a) FORTRAN.
    b) COBOL.
    c) C.
    d) LISP.

38. ¿Cuál de las operaciones siguientes no pertenece a un procesador de textos?

    a) Cursor arriba.
    b) Cursor abajo.
    c) Borrar una línea.
    d) Elevar un número al cuadrado.

39. Una base de datos simula un fichero manual pero con la ventaja de que:

    a) El cálculo es más rápido.
    b) Las fichas son más largas.
    c) La selección de fichas con alguna propiedad es más rápida.
    d) Las fichas se escriben tantas veces como queramos.

40. Una máquina de calcular potente y grande es el programa:

a) Procesador de textos.

b) Bases de datos.

c) BASIC.

d) Hoja electrónica.

SOLUCIONES DE LOS EJERCICIOS DE AUTOCOMPROBACIÓN

Capítulo 16

1. secuencia.
2. general
3. finito
4. ordenar
5. clasificación
6. selección
7. ordenar
8. adyacentes
9. ordenar
10. Shell
11. c)
12. b)
13. a)
14. b)
15. d)
16. c)
17. a)
18. d)
19. a)
20. c)
21. secuencial
22. cualquier
23. ordenadas
24. binaria
25. intercalación o Simple Merge
26. ordenadas
27. se llega al final de
28. en qué dia de la semana
29. bisiesto
30. MOD
31. c)
32. a)
33. a)
34. d)
35. a)
36. a)
37. d)
38. a)
39. b)
40. a)

Capítulo 17

1. b
2. a.
3. c.
4. a.
5. a.
6. c.
7. a.
8. b.
9. b.
10. c.
11. a.
12. almacenar.
13. directamente legible.
14. fichero.
15. jerárquico.
16. campo.
17. registro.
18. fichero.
19. bytes.
20. directorio.
21. secuencial.
22. directos.
23. indexados.
24. apertura.
25. registro.
26. cierre.
27. a.
28. b.
29. a.
30. c.
31. b.
32. a.
33. b.
34. a.
35. c.
36. a.
37. a.
38. reloj.
39. registros.
40. bits.
41. registro de instrucción.
42. acumulador.
43. principal o central.
44. dirección de memoria.
45. volátil.
46. bus.

Capítulo 18

1. sintaxis.
2. semántica.
3. semántica.
4. variables.
5. expresiones.
6. menor.
7. IF ... THEN.
8. función.
9. compilador.
10. reduce.
11. b.
12. d.
13. c.
14. c.
15. b.
16. d.
17. a.
18. c.
19. b.
20. d.
21. primitivos.
22. puntero.
23. uno.
24. registro.
25. registro.
26. menos.
27. control.
28. hoja electrónica.
29. desliga.
30. calcular.
31. b.
32. a.
33. d.
34. b.
35. d.
36. a.
37. d.
38. d.
39. c.
40. d.

Parte II

PRACTICAS CON EL ORDENADOR

Capítulo 16

ESQUEMA DE CONTENIDO

Aplicaciones diversas	Algoritmo de selección Algoritmo de burbuja Algoritmo de Shell Algoritmo de búsqueda lineal Búsqueda binaria Algoritmo de Simple Merge Algoritmo de Zeller Confección de un calendario Construcción de una agenda Un reloj digital Un reloj analógico

16.1 APLICACIONES DIVERSAS

La adaptación de los algoritmos de esta lección al ZX-SPECTRUM obliga a realizar unas pequeñas variaciones en algunos programas. Veamos cuáles son y téngalos presente a medida que va estudiándolos en el capítulo estándar.

16.1.1 Algoritmo de selección

En el programa de ordenación mediante el algoritmo de selección, añadiremos la línea:

```
56 CLS
```

modificaremos la 57:

```
57 PRINT AT 21,1 ;"Elemento ";I;
```

y suprimiremos la 210.

16.1.2 Algoritmo de burbuja

Lo mismo haremos en el programa de ordenación mediante el algoritmo de burbuja. Añadiremos:

```
72 CLS
```

modificaremos la 75:

```
75 PRINT AT 21,1;"Elemento: ";I;
```

y suprimiremos la 260.

16.1.3 Algoritmo de Shell

En el programa de ordenación con el algoritmo de Shell, añadiremos:

```
82 CLS
```

modificaremos la línea 85:

```
85 PRINT AT 21,1;"Elemento: ";I;
```

16.2.1

y suprimiremos la 310.

16.1.4 Algoritmo de búsqueda lineal

Para los procesos de búsqueda debemos también efectuar algunas modificaciones. En el programa de búsqueda lineal añadiremos:

```
82 CLS
```

modificaremos:

```
85 PRINT AT 21,1 ;" Elemento: ";I;
```

y suprimiremos la 210.

Si en este programa utilizamos una variable de cadena, recuerde que al dimensionarla deberá indicarle el número de caracteres que reserva.

16.1.5 Búsqueda binaria

En el de búsqueda binaria añadiremos:

```
82 CLS
```

modificamos la línea 85:

```
85 PRINT "Elemento: ";I;
```

y suprimiremos la línea 270.

16.1.6 Algoritmo de Simple Merge

En el algoritmo de intercalación por Simple Merge se modificarán las líneas:

```
 40 CLS : LET LP=20
 80 DIM A$(N,LP)
140 DIM B$(M,LP)
180 LET LR=N+M : DIM C$(LR,LP)
440 PRINT C$(I)(1 TO LP)
```

y se suprimirá la línea 460.

16.3

16.1.7 Algoritmo de Zeller

En el algoritmo de Zeller deben hacerse también algunos cambios, en las líneas siguientes:

```
 40 DIM M$(12,9) : DIM D$(7,9)
130 IF LEN(P$)<9 THEN P$=P$+" " : GOTO 130
150 IF I>12 THEN PRINT "MES ERRONEO" : PAUSE 10 : GOTO 120
220 LET C=VAL(A$(1 TO 2)) : LET Y=VAL(A$(3 TO 4))
315 STOP
```

16.1.8 Confección de un calendario

En el programa de confección del calendario, las modificaciones son:

```
  50 DIM M$(12,10) : DIM D$(7,4) : DIM D(12) : DIM N(12)
  65 LET R$="----------------------------"
  80 LET Y=VAL(A$(3 TO 4)) : LET C=VAL(A$(1 TO 2))
 265 PRINT " ";CD;" "
 350 STOP
4020 PRINT : PRINT TAB(7);M$(F);" ";A$
```

16.5

16.1.9 Construcción de una agenda

Vamos a ver a continuación algunos ejemplos de programas donde se utilizan los algoritmos que hemos visto, y que nos servirán además para repasar conceptos estudiados en otros capítulos.

El problema que se plantea es la construcción de una agenda. Pero en este caso no se trata de una agenda de direcciones, sino de una agenda de actividades. El proceso que tratamos de automatizar es el que efectuaría una persona muy ocupada, que escribe en su agenda el día y hora en que debe realizar cada actividad. (Por ejemplo, el día 4 a las 10 reunión del consejo, a las 12 comida de negocios, el día 6 a las 9 viaje al extranjero, etc.). Así, podrá saber si tiene un rato libre, o qué es lo que debe hacer a continuación.

Para automatizar este proceso, diseñaremos un programa en el cual podamos introducir las distintas actividades, indicando día y hora para realizarlas. El programa deberá controlar la colisión entre dos actividades (no se puede estar en dos sitios al mismo tiempo), y nos deberá indicar, cuando se lo preguntemos, las actividades que nos tocan en un día dado, escritas en el correspondiente orden cronológico.

Tal como lo planteamos, el programa deberá tener varias fases. Por una parte, habrá una entrada de datos, mediante la cual el usuario pueda ir introduciendo las diversas actividades a realizar. Por otra parte tendremos la fase de consulta. Cuando le preguntemos, repasará todos los datos introducidos, tomará los correspondientes al intervalo de tiempo que se pregunte, los ordenará convenientemente, y los escribirá en la pantalla. Finalmente, habrá un proceso de actualización, para borrar las actividades que ya se han realizado.

El proceso completo puede resultar algo complejo, dado que intervienen elementos diversos.

Así, al introducir cada dato deberá verificarse que la fecha que nos introducen es correcta. Esto supone que el número de días es el adecuado para el mes y el año que se escriba, es decir, no nos pueden escribir un número de días mayor que 31, y según en qué meses mayor que 30, o que 28 si se trata de febrero, excepto si el año es bisiesto.

El programa debe saber por tanto, el número máximo de días de cada mes de cada año, para un año cualquiera.

Por otra parte, y una vez hemos comprobado que la fecha escrita es correcta, deberemos controlar la hora que nos escriben para una actividad determinada, para evitar, tal como hemos dicho, la posible colisión de dos actividades. Pero puede darse el caso de que el usuario de la agenda desee cambiar una actividad por otra, el programa deberá permitir esta substitución, dando el aviso oportuno.

En cuanto a la fase de consulta, el programa deberá repasar todos los datos introducidos, y guardar y ordenar todos los que corresponden a la fecha buscada, para poderlos escribir. Para ello utilizaremos uno de los algoritmos de ordenación que hemos visto.

Finalmente, será necesaria una fase de borrado, para sacar de la agenda los datos que no tienen vigencia. Evidentemente, la información deja de tener interés cuando ha perdido actualidad; así, el día 10 de enero no nos importa en absoluto lo que teníamos apuntado para el día 9. Todo lo anterior a la fecha actual puede irse borrando. Para ello, lo único que haremos, será solicitar la fecha, y borrar toda la información anterior a ella.

El programa principal nos presentará las opciones posibles, en forma de menú. En la pantalla deberá aparecer este menú, que podría ser por ejemplo:

AGENDA

1. Entrada de actividades.
2. Consulta de la agenda.
3. Borrado de actividades.
4. FIN.

OPCIÓN:

El esquema general del proceso a seguir será:

Paso 1: Inicio.

Paso 2: Presentar menú.

Paso 3: Solicitar opción.

Paso 4: Si opción = finalizar ir al paso 10.

Paso 5: Si opción = entrada → Entrar datos.

Paso 6: Si opción = consulta → Consultar datos.

Paso 7: Si opción = borrado → Borrar datos.

Paso 8: Solicitar opción.

Paso 9: Volver a paso 4.

Paso 10: FIN.

Veamos ahora con más detalle los procesos de entrada de datos y de consulta.

Entrada de datos:

Paso 5.1: Inicio.

Paso 5.2: Solicitar fecha y comprobar si es correcta.

Paso 5.3: Solicitar hora, comprobar si es correcta, comprobar si hay colisión.

Paso 5.4: Solicitar actividad.

Paso 5.5: Almacenar los tres datos.

Consulta de datos:

Paso 6.1: Inicio.

Paso 6.2: Solicitar día de consulta.

Paso 6.3: Recorrer el almacén, seleccionando los datos correspondientes a ese día.

Paso 6.4: Ordenar.

Paso 6.5: Presentar el resultado en pantalla.

Borrado de datos

Paso 7.1: Inicio.

Paso 7.2: Solicitar fecha, y comprobar si es correcta.

Paso 7.3: Repasar todos los datos y borrar los anteriores a esta fecha.

Paso 7.4: Indicar el número de datos que quedan en la lista.

Una vez definido el proceso que ha de tener lugar, vamos a definir los datos que intervienen en el mismo, así como las variables que los van a contener.

Los datos de la agenda los almacenaremos en una tabla. Cada fila corresponderá a una actividad, en la primera columna se escribirá la fecha, en la segunda la hora y en la tercera la actividad.

El formato de la fecha será DD-MM-AA, es decir, dos dígitos para el día, un guión, dos dígitos para el mes, otro guión, y dos dígitos para el año. El formato para la hora será HH:MM, es decir, dos dígitos para la hora, dos puntos, y dos dígitos para los minutos.

Estos dos formatos han de ser fijos, pues de lo contrario no podríamos realizar correctamente la ordenación.

La tabla estará contenida en la variable *D$*, que dimensionaremos al principio del programa, a un valor máximo, por ejemplo de 50 actividades. El número total de elementos de esta tabla será pues de 50 x 3. Definiremos por otra parte un contador que se irá incrementando cada vez que escribamos una nueva actividad, y cuyo valor evidentemente no podrá superar este máximo.

Escribiremos ahora las distintas partes del programa. Tal como hemos hecho en el capítulo estándar, resolveremos los distintos problemas por separado, y cuando hayamos comprobado que funcionan correctamente, los convertiremos en subrutinas del programa principal.

El primer problema que vamos a plantearnos es la validación de la fecha; es decir, un sistema que indique si una fecha cualquiera es o no correcta. Para ello utilizaremos los mecanismos que hemos visto en la confección del calendario. Así, cuando nos introduzcan una fecha cualquiera, en el formato que anteriormente hemos indicado, deberemos separar día, mes y año, y comprobar si poseen un valor correcto. Veamos cuál sería el listado del programa para ello.

```
  10 REM ==================================
  20 REM Validacion de una fecha cualquiera
  30 REM ----------------------------------
  50   DIM d(12)
  60   GOSUB 5060
2000 REM ---- Entrar fecha ----
2010   CLS : INPUT "Entre la fecha: ";f$
2020   IF LEN(f$)<>8 THEN GOTO 2010
2030   LET d$ = f$(1 TO 2)
```

```
2040    LET m$ = f$(4 TO 5)
2050    LET a$ = f$(7 TO 8)
2060    IF d$<"00" OR d$>"31" OR m$<"00" OR m$>"12"
        OR a$<"00" OR a$>"99" THEN GOTO 2130
2080    LET dd = VAL (d$) : LET mm = VAL (m$)
2090    LET  a = VAL("19"+a$) : GOSUB 5000
2100    IF  dd>d(mm) THEN GOTO 2130
2110    PRINT "Fecha correcta"
2120   STOP
2130   PRINT "Fecha erronea"
2140   PAUSE 50 : GOTO 2010
5000 REM ---- Mes de febrero ----
5010     LET r1 = a-INT(a/4)*4
5020     LET r2 = a-INT(a/100)*100
5030     LET r3 = a-INT(a/400)*400
5040     LET d(2) = 28
5050     IF r1=0 AND r2<>0 AND r3<>0 THEN LET d(2)=29
5055    RETURN
5060 REM ---- Restantes meses ----
5070     LET d(1) = 31
5080     FOR i = 3 TO 12
5085         READ d(i)
5090         NEXT i
5095     RETURN
9000 REM ---- Almacen de datos ----
9010     DATA 31,30,31,30,31,31,30,31,30,31
```

Una vez hemos obtenido una fecha correcta, se deberá introducir y comprobar la hora, para lo cual deberemos separar los dos primeros dígitos y los dos últimos introducidos; el valor de los primeros debe ir de 00 a 23, el de los dos últimos de 00 a 59.

Antes de almacenar los valores entrados, hay que verificar, tal como hemos dicho, que no existe colisión. Si no la hay, se podrá efectuar la entrada directamente; si la hay, se deberá dar el correspondiente aviso al usuario, para que confirme la entrada realizada, en cuyo caso la nueva actividad substituirá a la antigua. Si el usuario no da la confirmación, se volverá a solicitar una nueva fecha y hora. Puede probar el programa escribiendo RUN y dándole una fecha según se ha indicado.

Por último, se deberá introducir la actividad, sobre la cual no se hará ninguna comprobación.

Con esto ya tendremos los tres datos entrados. La entrada se hará sin embargo sobre una variable auxiliar. Cuando se haya comprobado que los valores son correctos, se irán pasando a la tabla definitiva. Veamos el programa completo para la entrada de datos de la agenda.

```
  10 REM ==================================
  20 REM AGENDA  Entrada de datos
  30 REM ----------------------------------
  40   LET mx=50 : LET na=0
  50   DIM d(12) : DIM j$(mx,3,40)
  60   GOSUB 5060 : REM Dias de los meses
  70   GOSUB 2000 : REM Entrar la fecha
  80   GOSUB 2500 : REM Entrar la hora
  90   GOSUB 2700 : REM Comprobar colision
```

```
  95   REM             y almacenar actividad
 100   INPUT "Desea continuar (S/N): ";r$
 110   IF r$="S" OR r$="s" THEN GOTO 70
 120 STOP
2000 REM ---- Entrar fecha ----
2010   CLS : INPUT "Entre la fecha: ";f$
2020   IF LEN(f$)<>8 THEN GOTO 2010
2030   LET d$ = f$(1 TO 2)
2040   LET m$ = f$(4 TO 5)
2050   LET a$ = f$(7 TO 8)
2060   IF d$<"00" OR d$>"31" OR m$<"00" OR m$>"12"
       OR a$<"00" OR a$>"99" THEN GOTO 2130
2080   LET dd = VAL (d$) : LET mm = VAL (m$)
2090   LET  a = VAL("19"+a$) : GOSUB 5000
2100   IF  dd>d(mm) THEN GOTO 2130
2110   GOTO 2150
2130  PRINT "Fecha erronea"
2140  PAUSE 50 : GOTO 2010
2150 RETURN
2500 REM ---- Entrar hora ----
2510    INPUT "Entre la hora: ";h$
2515    IF LEN(h$)<>5 THEN GOTO 2510
2520    LET i$=h$(1 TO 2) : LET m$=h$(4 TO 5)
2530   IF i$<"00" OR i$>"23" THEN GOTO 2510
2540   IF m$<"00" OR m$>"59" THEN GOTO 2510
2550   RETURN
2700 REM ---- Comprobar colision ----
2710   LET i=1 : LET rs=0
2720   IF i>na OR rs THEN GOTO 2780
2730      IF j$(i,1)(1 TO 8)<>f$ THEN GOTO 2760
2740      IF j$(i,2)(1 TO 5)<>h$ THEN GOTO 2760
2750      LET rs = 1
2755      GOTO 2720
2760      LET i=i+1
2770      GOTO 2720
2780   IF i>na THEN GOTO 2840
2785 REM  ---- Colision ----
2790      PRINT "El dia ";f$;" a las ";h$
2800      PRINT "tiene programado:" : PRINT j$(i,3)
2810      PRINT : INPUT "Desea modificarlo (s/n): ";r$
2820      IF r$="N" OR r$="n" THEN GOTO 2920
2825 REM ---- Cambio ----
2830   LET pa=i : GOTO 2860
2840 REM ---- Nueva actividad ----
2850   IF na=mx THEN GOTO 2900
2855 REM ---- Almacenamiento ----
2860   LET na=na+1 : LET pa=na
2870      LET j$(pa,1)=f$ : LET j$(pa,2)=h$
2880      INPUT "Escriba la actividad: ";j$(pa,3)
2890      GOTO 2920
2895 REM ---- Agenda completa ----
2900   PRINT "La agenda esta llena"
2910   PRINT "No puede escribir mas actividades"
2920   RETURN
5000 REM ---- Mes de febrero ----
5010    LET r1 = a-INT(a/4)*4
5020    LET r2 = a-INT(a/100)*100
5030    LET r3 = a-INT(a/400)*400
5040    LET d(2) = 28
5050    IF r1=0 AND r2<>0 AND r3<>0 THEN LET d(2)=29
```

```
5055 RETURN
5060 REM ---- Restantes meses ----
5070     LET d(1) = 31
5080     FOR i = 3 TO 12
5085         READ d(i)
5090         NEXT i
5095     RETURN
9000 REM ---- Almacen de datos ----
9010     DATA 31,30,31,30,31,31,30,31,30,31
```

Este programa nos permitirá ir escribiendo actividades en nuestra agenda, mientras indiquemos que deseamos continuar.

Los comentarios que se incluyen en los distintos apartados van indicando las sucesivas etapas del programa. Así, en primer lugar se dimensiona la tabla que ha de contener los datos almacenados, y la lista que contiene el número de días de cada mes (línea 50). Esta lista se lee en la subrutina 5060.

Se procede entonces a entrar la fecha, cuya validación es la que hemos visto anteriormente, aunque en este caso se realiza dentro de una subrutina (líneas 2.000 a 2.150). Para ello se borra la línea 2.110, se añade:

```
2120 GOTO 2150
```

De esta forma, el programa da únicamente un mensaje en el caso de que la fecha introducida sea errónea, solicitando que se entre de nuevo. Si la fecha es correcta se produce directamente el retorno al programa principal.

Se pasa entonces a entrar la hora, dentro de la subrutina 2.500, donde también se controla que sea correcta.

Por último, tal como ya hemos dicho, hay que comprobar la posibilidad de colisión. Esto se hace en la subrutina 2.700. En ella se deben contemplar varias posibilidades:

- Al repasar la agenda ya existe una actividad para igual fecha y hora. En este caso hay que preguntar al usuario, y sustituir la actividad almacenada por la nueva si así lo desea. En caso contrario, no se modificará la tabla.
- No hay colisión, pero la agenda está completa. En este caso se dará un mensaje al usuario; la tabla no se modificará, dado que ya no caben más elementos. A patir de ahora y mientras no se borre alguna información, únicamente podrán cambiarse las actividades existentes por otras, pero no se podrá añadir ninguna.
- No hay colisión y la tabla no está completa. En este caso se añadirá al final de la lista la actividad entrada.

Para saber si hay colisión se repasa toda la agenda. Se comprueba en primer lugar si la fecha entrada es distinta de la que se lee en la lista (línea 2.730), si no es así, se leerá el siguiente elemento de la misma, hasta encontrar coincidencia, o llegar al final de la lista. Si fuera la fecha igual hay que comprobar si coinciden las horas (línea 2.740). Si también son iguales, se modifica el valor de la variable *rs*, con lo que saldremos del bucle, dado que tal como está construido, se repetirá mientras no se haya recorrido toda la tabla (mientras *i* sea menor que *na*), y simultáneamente *rs* sea igual a 0. Así, cuando existe coincidencia de fecha y hora, asignamos a *rs* el valor 1, con lo que salimos del bucle.

Se dará entonces la indicación de que se ha producido colisión (líneas 2.790 a 2.810), preguntando asimismo al usuario si desea o no modificar la actividad. Si la respuesta es negativa, no hay que modificar la lista, y podremos pasar a escribir una nueva actividad. Si por el contrario, queremos variar lo que habíamos escrito, pasamos a entrar la nueva actividad, situándola en la posición de la anterior en la tabla.

Por otra parte, si no hay coincidencia en fecha y hora, podrá escribirse la actividad. Esta se almacenará en la última posición de la tabla, después de haber comprobado que esta posición es todavía inferior o igual al número máximo de elementos fijado.

Ahora se preguntará al usuario que desea continuar, con lo que se repetirá el proceso, entrando nuevas actividades, o modificando las existentes, mientras el usuario lo desee, siempre que no se haya llegado a llenar toda la tabla.

Con esto hemos completado la fase de entrada de datos. Veamos ahora el proceso de consulta. Para ello supondremos que hemos escrito una serie de actividades en la agenda (el número está contenido en la variable *na*). El usuario deberá dar la fecha del día del cual desea conocer las actividades que tiene programadas. El programa controlará que la fecha escrita sea correcta. Para ello utilizaremos la misma subrutina que en el caso anterior (la número 2.000).

Para efectuar la selección de las actividades de cada fecha, el programa deberá recorrer la lista completa, seleccionar las correspondientes a ese día, guardándolas en una nueva lista, para posteriormente ordenarlas cronológicamente, y presentarlas al usuario.

Dado que se seleccionan todas las actividades de una misma fecha, no hará falta que la vayamos almacenando. Así, la nueva tabla tendrá sólo dos columnas, la de hora y la de actividad.

Añadiremos pues las rutinas de consulta, ordenación y presentación de resultado, cuyos listados son los siguientes:

```
3000 REM ---- Consulta ----
3010    GOSUB 2000 : REM Entrada de la fecha a consultar
3050    LET is = 0
3060    FOR i=1 TO na
3070       IF f$<>j$(i,1)(1 TO 8)  THEN GOTO 3120
3080       LET is=is+1
3090       LET s$(is,1)=j$(i,2)
```

```
3100        LET s$(is,2)=j$(i,3)
3120        NEXT i
3130     RETURN
4000 REM ---- Ordenacion de  S$ con IS elementos ----
4010 REM ---- Algoritmo de seleccion  ----
4020     FOR i=1 TO is-1
4030       FOR j=i+1 TO is
4040         IF s$(i,1)<s$(j,1) THEN GOTO 4140
4050         LET i$=s$(i,1)
4060         LET o$=s$(i,2)
4080         LET s$(i,1)=s$(j,1)
4090         LET s$(i,2)=s$(j,2)
4110         LET s$(j,1)=i$
4120         LET s$(j,2)=o$
4140         NEXT j
4150      NEXT i
4160 RETURN
5500 REM ---- Resultado ----
5505     CLS : PRINT "Actividades del dia ";f$ : PRINT
5510     FOR i=1 TO is
5560     PRINT s$(i,1)(1 TO 6);s$(i,2)
5570     NEXT i
5580 RETURN
```

Después de entrar y comprobar la fecha (en la subrutina 2.000), se realizará la selección de los datos, que se almacenan en la tabla *s$* (subrutina 3.000). Observe que para ello comprobamos si coinciden únicamente los elementos de la primera columna de la tabla (la fecha). Si no es así, seguimos la exploración; si es así, incrementamos el contador de elementos seleccionados (*is*), y almacenamos este elemento. Tal como hemos dicho, hace falta únicamente guardar hora y actividad. Evidentemente será necesario dimensionar esta tabla; añadiremos pues a la línea 50:

```
DIM s$(mx,2,40)
```

Por otra parte, para ordenar esta tabla seguimos un algoritmo ya conocido, el de selección, aunque en este caso existe una diferencia, dado que ordenamos una tabla, no una lista.

En este caso, nos guiamos por la primera columna, que contiene las horas, sus elementos serán los que compararemos de cara a ordenar; cada vez deberemos intercambiar las dos columnas, hora y actividad.

Veamos ahora la última fase: borrado de las actividades obsoletas.

Para ello el usuario deberá entrar una fecha. Todas las actividades anteriores a ella se borrarán de la lista. El proceso efectivo de borrado lo efectuaremos sobre la propia tabla de datos. Para ello empezaremos ordenando la tabla en orden decreciente de fechas. Una vez ordenada, buscaremos en la primera columna la fecha a partir de la cual debemos

borrar. Dado que la tabla está organizada en orden decreciente de fechas, podremos asegurar que todas las actividades que están a partir de ella corresponden a días anteriores al que hemos indicado. Por tanto, son las que se desean borrar. Asignaremos a la variable *na* el valor anterior al que tiene el primer elemento cuya fecha sea menor o igual que la entrada por el usuario, con lo que a partir de este momento, y aunque no hayamos hecho la acción de borrar, todas estas actividades han dejado de estar a efectos prácticos. El hecho de trasladar el valor de *na* equivale pues a suprimir todos los elementos a partir de este punto: la tabla va de 1 a *na*. Si *na* vale 10, por ejemplo, tenemos diez elementos en la lista para buscar o modificar. Si ahora cambiamos este valor a 6 por ejemplo, sólo tendremos 6 elementos; aunque los restantes cuatro estén en la lista, observe que todos los procesos se efectúan según el valor de *na*, ya que buscamos hasta *na*, cuando entramos un elemento nuevo lo hacemos en la posición *na*, etc., de manera que hemos prescindido de los elementos de la lista situados a partir de *na*.

Veamos el listado de la subrutina para la operación de borrado.

```
6000 REM ---- Ordenacion ----
6010   FOR i=1 TO na-1
6020      FOR j=i+1 TO na
6030         LET i$=j$(i,1)(1 TO 2)
6035         LET o$=j$(j,1)(1 TO 2)
6040         LET k$=j$(i,2)(4 TO 5)
6045         LET l$=j$(j,2)(4 TO 5)
6050         LET m$=j$(i,3)(7 TO 8)
6055         LET n$=j$(j,3)(7 TO 8)
6060         IF m$>n$ THEN GOTO 6150
6070         IF m$=n$ AND  k$>l$ THEN GOTO 6150
6080         IF m$=n$ AND k$=l$ AND i$>o$ THEN GOTO 6150
6082         LET i$=j$(i,1)
6084         LET o$=j$(i,2)
6086         LET k$=j$(i,3)
6090         LET j$(i,1)=j$(j,1)
6100         LET j$(i,2)=j$(j,2)
6110         LET j$(i,3)=j$(j,3)
6120         LET j$(j,1)=i$
6130         LET j$(j,2)=o$
6140         LET j$(j,3)=k$
6150         NEXT j
6160      NEXT i
6170    RETURN
6500 REM Repaso de la tabla y actualizacion
6510   LET i=1
6520   IF j$(i,1)(1 TO 8)<=f$ THEN GOTO 6550
6530      LET i=i+1
6540      GOTO 6520
6550   LET na=i-1
6555   CLS : PRINT "En la tabla quedan ";na;" elementos"
6560 RETURN
```

Para el proceso de ordenación hemos utilizado también el algoritmo de selección. En este caso sin embargo, el proceso es algo más complejo, ya que debemos ordenar por la primera columna de la tabla, es decir por fechas. Observe que para comparar dos fechas cualquiera, por ejem-

plo: 30-10-84 con 10-09-85, no podemos hacerlo directamente, ya que siguiendo el procedimiento de comparación de la máquina –de izquierda a derecha–, sería mayor la fecha del 30-10-84 que la del 10-09-85, siendo evidentemente posterior la segunda.

Por ello debemos hacer la comparación al revés, considerando primero el año, después el mes y por último el día, tal como hacemos en el listado.

Por otra parte, al efectuar los intercambios, dado que se trata de una tabla, deberemos cambiar la posición de los tres elementos, la fecha, la hora y la actividad.

Ahora, una vez tengamos la tabla ordenada en orden decreciente de fechas, la empezaremos a recorrer por el principio. Mientras la fecha leída sea superior a la que nos ha indicado el usuario, dejaremos la tabla igual. Cuando encontremos una fecha igual o inferior al valor entrado, desplazaremos el puntero que indica el fin de la tabla (na) a la posición inmediatamente anterior a esa fecha. Con ello, tal como hemos dicho, suprimimos a efectos prácticos todos los elementos restantes.

En este momento tenemos construidas todas las subrutinas que van a intervenir en el programa completo. Nos quedará ahora únicamente escribir el programa principal, que ha de presentar en pantalla el menú, controlar la utilización global del programa y añadir algunas líneas para encajar el programa completo. Repase, por tanto, su listado con el que le presentamos a continuación.

```
 10 REM ====================================
 20 REM AGENDA DE ACTIVIDADES
 30 REM ------------------------------------
 40   LET mx=50  : LET na=0
 50    DIM d(12) : DIM j$(mx,3,40)  : DIM s$(mx,2,40)
 54    GOSUB 5060 : REM Dias de cada mes
 55 REM ---- Menu ----
 56    CLS : PRINT : PRINT : PRINT
 57    PRINT TAB(5);"1.- Entrar actividades" : PRINT
 58    PRINT TAB(5);"2.- Consulta" : PRINT
 60    PRINT TAB(5);"3.- Borrado actividades" : PRINT
 62    PRINT TAB(5);"4.- FIN"
 64    PRINT : PRINT : PRINT
 68 REM ---- Pedir opcion ----
 70    INPUT "OPCION: ";p$
 72    IF p$<>"1" THEN GOTO 76
 74       GOSUB 2000 : GOSUB 2500 : GOSUB 2700 : GOTO 98
 76    IF p$<>"2" THEN GOTO 80
 78       GOSUB 3000 : GOSUB 4000 : GOSUB 5500 : GOTO 98
 80    IF p$<>"3" THEN GOTO 84
 82       GOSUB 2000 : GOSUB 6000 : GOSUB 6500 : GOTO 98
 84    IF p$<>"4" THEN GOTO 56
 86       STOP
 98    IF INKEY$="" THEN GOTO 98
100    GOTO 56
```

Con esto tendremos completo el programa de manipulación de la agenda. El listado completo será pues:

```
  10 REM ====================================
  20 REM AGENDA DE ACTIVIDADES
  30 REM ------------------------------------
  40   LET mx=50  : LET na=0
  50    DIM d(12) : DIM j$(mx,3,40) : DIM s$(mx,2,40)
  54    GOSUB 5060 : REM Dias de cada mes
  55 REM ---- Menu ----
  56    CLS : PRINT : PRINT : PRINT
  57    PRINT TAB(5);"1.- Entrar actividades" : PRINT
  58    PRINT TAB(5);"2.- Consulta" : PRINT
  60    PRINT TAB(5);"3.- Borrado actividades" : PRINT
  62    PRINT TAB(5);"4.- FIN"
  64    PRINT : PRINT : PRINT
  68 REM ---- Pedir opcion ----
  70    INPUT "OPCION: ";p$
  72    IF p$<>"1" THEN GOTO 76
  74       GOSUB 2000 : GOSUB 2500 : GOSUB 2700 : GOTO 98
  76    IF p$<>"2" THEN GOTO 80
  78       GOSUB 3000 : GOSUB 4000 : GOSUB 5500 : GOTO 98
  80    IF p$<>"3" THEN GOTO 84
  82       GOSUB 2000 : GOSUB 6000 : GOSUB 6500 : GOTO 98
  84    IF p$<>"4" THEN GOTO 56
  86       STOP
  98    IF INKEY$="" THEN GOTO 98
 100    GOTO 56
2000 REM ---- Entrar fecha ----
2010   CLS : INPUT "Entre la fecha: ";f$
2020   IF LEN(f$)<>8 THEN GOTO 2010
2030   LET d$ = f$(1 TO 2)
2040   LET m$ = f$(4 TO 5)
2050   LET a$ = f$(7 TO 8)
2060   IF d$<"00" OR d$>"31" OR m$<"00" OR m$>"12"
       OR a$<"00" OR a$>"99" THEN GOTO 2130
2080   LET dd = VAL (d$) : LET mm = VAL (m$)
2090   LET  a = VAL("19"+a$) : GOSUB 5000
2100   IF  dd>d(mm) THEN GOTO 2130
2110   GOTO 2150
2130  PRINT "Fecha erronea"
2140  PAUSE 50 : GOTO 2010
2150 RETURN
2500 REM ---- Entrar hora ----
2510    INPUT "Entre la hora: ";h$
2515    IF LEN(h$)<>5 THEN GOTO 2510
2520    LET i$=h$(1 TO 2) : LET m$=h$(4 TO 5)
2530   IF i$<"00" OR i$>"23" THEN GOTO 2510
2540   IF m$<"00" OR m$>"59" THEN GOTO 2510
2550   RETURN
2700 REM ---- Comprobar colision ----
2710   LET i=1 : LET rs=0
2720   IF i>na OR rs THEN GOTO 2780
2730      IF j$(i,1)(1 TO 8)<>f$ THEN GOTO 2760
2740      IF j$(i,2)(1 TO 5)<>h$ THEN GOTO 2760
2750      LET rs = 1
2755      GOTO 2720
2760      LET i=i+1
2770      GOTO 2720
2780   IF i>na THEN GOTO 2840
2785 REM  ---- Colision ----
2790      PRINT "El dia ";f$;" a las ";h$
2800      PRINT "tiene programado:" : PRINT j$(i,3)
2810      PRINT : INPUT "Desea modificarlo (s/n): ";r$
2820      IF r$="N" OR r$="n" THEN GOTO 2920
2825 REM ---- Cambio ----
2830   LET pa=i : GOTO 2860
2840 REM ---- Nueva actividad ----
2850   IF na=mx THEN GOTO 2900
2855 REM ---- Almacenamiento ----
2860   LET na=na+1 : LET pa=na
2870      LET j$(pa,1)=f$ : LET j$(pa,2)=h$
2880      INPUT "Escriba la actividad: ";j$(pa,3)
2890      GOTO 2920
2895 REM ---- Agenda completa ----
2900   PRINT "La agenda esta llena"
2910   PRINT "No puede escribir mas actividades"
2915   PRINT "Debe borrar"
2920   RETURN
```

```
3000 REM ---- Consulta ----
3010    GOSUB 2000 : REM Entrada de la fecha a consultar
3050    LET is = 0
3060    FOR i=1 TO na
3070       IF f$<>j(i,1)(1 TO 8)  THEN GOTO 3120
3080       LET is=is+1
3090       LET s$(is,1)=j$(i,2)
3100       LET s$(is,2)=j$(i,3)
3120       NEXT i
3130    RETURN
4000 REM ---- Ordenacion de  S$ con IS elementos ----
4010 REM ---- Algoritmo de seleccion  ----
4020    FOR i=1 TO is-1
4030      FOR j=i+1 TO is
4040        IF s$(i,1)<s$(j,1) THEN GOTO 4140
4050        LET i$=s$(i,1)
4060        LET o$=s$(i,2)
4080        LET s$(i,1)=s$(j,1)
4090        LET s$(i,2)=s$(j,2)
4110        LET s$(j,1)=i$
4120        LET s$(j,2)=o$
4140        NEXT j
4150     NEXT i
4160 RETURN
5000 REM ---- Mes de febrero ----
5010    LET r1 = a-INT(a/4)*4
5020    LET r2 = a-INT(a/100)*100
5030    LET r3 = a-INT(a/400)*400
5040    LET d(2) = 28
5050    IF r1=0 AND r2<>0 AND r3<>0 THEN LET d(2)=29
5055 RETURN
5060 REM ---- Restantes meses ----
5070    LET d(1) = 31
5080    FOR i = 3 TO 12
5085        READ d(i)
5090       NEXT i
5095    RETURN
5500 REM ---- Resultado ----
5505    CLS : PRINT "Actividades del dia ";f$ : PRINT
5510    FOR i=1 TO is
5560    PRINT s$(i,1)(1 TO 6);s$(i,2)
5570    NEXT i
5580 RETURN
6000 REM ---- Ordenacion ----
6010   FOR i=1 TO na-1
6020      FOR j=i+1 TO na
6030         LET i$=j$(i,1)(1 TO 2)
6035         LET o$=j$(j,1)(1 TO 2)
6040         LET k$=j$(i,2)(4 TO 5)
6045         LET l$=j$(j,2)(4 TO 5)
6050         LET m$=j$(i,3)(7 TO 8)
6055         LET n$=j$(j,3)(7 TO 8)
6060         IF m$>n$ THEN GOTO 6150
6070         IF m$=n$ AND  k$>l$ THEN GOTO 6150
6080         IF m$=n$ AND k$=l$ AND i$>o$ THEN GOTO 6150
6082         LET i$=j$(i,1)
6084         LET o$=j$(i,2)
6086         LET k$=j$(i,3)
6090         LET j$(i,1)=j$(j,1)
6100         LET j$(i,2)=j$(j,2)
6110         LET j$(i,3)=j$(j,3)
6120         LET j$(j,1)=i$
6130         LET j$(j,2)=o$
6140         LET j$(j,3)=k$
6150         NEXT j
6160      NEXT i
6170    RETURN
6500 REM Repaso de la tabla y actualizacion
6510   LET i=1
6520   IF j$(i,1)(1 TO 8)<=f$ THEN GOTO 6550
6530      LET i=i+1
6540      GOTO 6520
6550   LET na=i-1
6555   CLS : PRINT "En la tabla quedan ";na;" elementos"
6560 RETURN
9000 REM ---- Almacen de datos ----
9010    DATA 31,30,31,30,31,31,30,31,30,31
```

16.1.10 Un reloj digital

Vamos a ver ahora cómo construir un reloj digital. En capítulos anteriores ya vimos el algoritmo básico para ello. Sin embargo ahora, con todo lo que hemos aprendido vamos a mejorarlo, de forma que podamos poner el reloj a la hora que deseemos, y por otra parte, nos presente los dígitos correspondientes a horas y minutos de tamaño grande, aprovechando también lo que hemos visto sobre gráficos en la lección correspondiente.

El listado que habíamos visto para construir un reloj es el siguiente:

```
NEW
10 FOR I = 0 TO 23
20     FOR J = 0 TO 59
30        FOR K = 0 TO 59
40           CLS
50           PRINT I;":";J;":";K;
60           FOR L = 1 TO 100
70              NEXT L
80           NEXT K
90        NEXT J
100    NEXT I
```

Tal como está el programa, el reloj siempre empezará a partir de las 0 horas 0 minutos 0 segundos, o lo que es lo mismo, a las 12 en punto de la noche. Por tanto, si deseáramos utilizarlo para saber la hora que es, nos veríamos obligados a poner el programa en marcha la noche del día anterior. Se hace pues evidente la necesidad de poder ponerlo en hora, en el momento que lo pongamos en marcha.

Es posible que le llame la atención la línea 60. Su finalidad es entretener el ordenador durante un segundo, antes de pasar al siguiente valor de *K*; es decir, que entre el momento en que el reloj marca la hora (intrucción PRINT en la línea 50), y el momento en que vuelva a escribirla (el valor de K habrá aumentado en una unidad), haya transcurrido efectivamente un segundo de tiempo. Usted puede tratar de ajustarlo cambiando el valor final del bucle (100) hasta conseguir que el reloj vaya al mismo ritmo que el que usted tiene.

Para poner el reloj en funcionamiento, el programa deberá empezar preguntando horas, minutos y segundos, y empezar a contar a partir de aquí. Por tanto, habrá que añadir al listado:

```
2 CLS
4 INPUT "HORA: ";H1
6 INPUT "MINUTOS: ";M1
8 INPUT "SEGUNDOS: ";S1
```

Naturalmente, habrá también que modificar el programa de forma que los bucles empiecen a partir de estos valores de hora, minutos y segundos. Deberemos por tanto modificar las líneas:

```
10 FOR I = H1 TO 23
20     FOR J=M1 TO 59
30         FOR K=S1 TO 59
```

De esta forma, el reloj empezará a contar a partir de la hora que hemos escrito. Esto irá bien para la primera vuelta. ¿Pero y las siguientes? ¿Qué pasará cuando haya llegado a escribir los 59 segundos y tenga que saltar al minuto siguiente?

Tal como hemos visto, el bucle interior se volverá a ejecutar desde el principio, pero ahora el principio no está en cero, tal como debería, sino en el valor que hemos entrado como segundo inicial. Así, suponga que hemos entrado los valores siguientes de hora, minutos y segundos:

HORA: 10

MINUTOS: 22

SEGUNDOS: 58

El reloj empezará escribiendo:

10:22:58

Esperará un segundo y escribirá a continuación:

10:22:59

Ahora habrá terminado el bucle más interior, ya que K ha llegado a 59. Se incrementará el contador de minutos –valdrá 23– y volverá a iniciarse el bucle interior. Pero tal como indica el listado, este bucle empieza en S1, es decir en 58. El reloj marcará a continuación:

10:23:58

Evidentemente, su funcionamiento no es correcto. Cuando ha terminado la primera vuelta completa, para cada bucle, hay que modificar el valor de inicio de cada uno, de forma que a partir de la primera vuelta, en que el reloj empieza a marcar la hora que nosotros le indicamos, el valor de H1, M1 y S1 vuelva a 0 para que el funcionamiento sea el correcto.

Así, habrá que añadir las líneas:

```
 85          LET S1=0
 95     LET M1=0
110 LET H1=0
120 GOTO 10
```

Tal como teníamos escrito el listado, el reloj funcionaría a lo largo de un día, hasta completar todos los bucles, y el programa se detendría. Ahora, añadiendo la línea 120, cuando acabe un día –a las 23:59:59– volverá a empezar a ejecutarse el primer bucle, lo que equivale a que el reloj seguirá funcionando, sin detenerse, hasta que forcemos la interrupción del programa pulsando la tecla correspondiente.

El listado completo quedará pues:

```
  2 CLS
  4 INPUT "HORA: ";H1
  6 INPUT "MINUTOS: ";M1
  8 INPUT "SEGUNDOS: ";S1
 10 FOR I = H1 TO 23
 20     FOR J = M1 TO 59
 30         FOR K = S1 TO 59
 40             CLS
 50             PRINT I;":";J;":";K;
 60             FOR L = 1 TO 100
 70                 NEXT L
 80             NEXT K
 85         LET S1 = 0
 90         NEXT J
 95     LET M1 = 0
100     NEXT I
110 LET H1=0
120 GOTO 10
```

Con esto hemos conseguido la primera mejora que pretendíamos.

Vamos a ver cómo hacemos la segunda. Se trata, tal como hemos dicho, de que el reloj escriba las cifras correspondientes a horas y minutos, de tamaño grande.

En un capítulo anterior, hemos visto un programa para construir números grandes. Este procedimiento lo añadiremos en forma de subrutina al empezar nuestro programa, de forma que éste nos represente los caracteres correspondientes a cada cifra. Para ello, añadiremos o modificaremos las instrucciones:

```
   2 CLS :  DIM A$(10,3,2)
   3 GOSUB 500 : GOSUB 1000
 500 REM Inicio
 510 LET XH = 1 : LET XD = 10
 520 LET XS = 20 : LET YS = 3
 530 RETURN
1000 REM Lectura de los caracteres de cada cifra
1010     FOR I=1 TO 10
1020         FOR J=1 TO 3
1030             READ G,R : LET A$(I,J)=CHR$(128+G)+CHR$(128+R)
1040             NEXT J
1050         NEXT I
1060     RETURN
```

```
2000 DATA 11,7,10,5,14,13
2010 DATA 9,10,0,10,4,14
2020 DATA 3,7,4,2,14,12
2030 DATA 3,7,4,13,12,13
2040 DATA 10,0,14,14,0,10
2050 DATA 11,3,14,12,12,13
2060 DATA 11,3,14,12,14,13
2070 DATA 3,7,0,13,0,5
2080 DATA 11,7,15,15,14,13
2090 DATA 11,7,14,13,0,5
```

El programa empezará leyendo las posiciones de las cifras (para las horas y los minutos) en la subrutina 500, así como los caracteres correspondientes a cada cifra, en la subrutina 1.000, para escribirlas en lugar de los números correspondientes a horas y minutos. Dado que la máquina invierte un cierto tiempo en el cálculo y representación de estos caracteres, únicamente cambiaremos las cifras de horas y minutos; los segundos los escribiremos directamente, para evitar el posible retraso debido a la transformación.

La escritura de horas y minutos habrá que hacerla pues en los bucles más exteriores; el programa deberá modificarse de la siguiente forma:

```
50              PRINT AT YS,XS ;K;" "
13   CLS
14   IF I<10 THEN LET HI=1 : LET HD=I+1 : GOTO 16
15   LET H$=STR$(I) : LET HI=VAL(H$(1 TO 1))+1 :
LET HD=VAL(H$(2 TO 2))+1
16   FOR  N=1 TO 3
17      PRINT AT N,XH ;A$(HI,N);" ";A$(HD,N)
18      NEXT N
23   IF J<10 THEN LET MI=1 : LET MD=J+1 : GOTO 25
24   LET M$=STR$(J) : LET MI=VAL(M$(1 TO 1))+1 :
LET MD=VAL(M$(2 TO 2))+1
25   FOR  N=1 TO 3
26      PRINT AT N,XD;A$(MI,N);" ";A$(MD,N)
27      NEXT N
```

Habrá que borrar además, la línea 40. Observe además, que se ha modificado la forma de construir los caracteres de cara a facilitar su escritura. En este caso, cada cifra se construye con seis caracteres, tal como hacíamos antes, pero almacenándolos de dos en dos, de forma que para escribir cada cifra deberemos escribir tres líneas, de dos caracteres cada vez. Por otra parte, para posicionar estos símbolos se utilizan unas variables determinadas: *YS*, *XS*, *XD*, y *XH*, cuyos valores se asignarán, tal como hemos visto, en la subrutina 500.

El listado completo del programa será:

```
   1 REM    ---  RELOJ DIGITAL  ---
   2 CLS : DIM A$(10,3,2)
   3 GOSUB 500 : GOSUB 1000
   4 INPUT "HORA: ";H1
   6 INPUT "MINUTOS: ";M1
   8 INPUT "SEGUNDOS: ";S1
  10 FOR I= H1 TO 23
  13   CLS
  14   IF I<10 THEN LET HI=1 : LET HD=I+1 : GOTO 16
  15   LET H$=STR$(I) : LET HI=VAL(H$(1 TO 1))+1 :
 LET HD=VAL(H$(2 TO 2))+1
  16   FOR N=1 TO 3
  17      PRINT AT N,XH; A$(HI,N);" ";A$(HD,N)
  18      NEXT N
  20   FOR J=M1 TO 59
  23      IF J<10 THEN LET MI=1 : LET MD=J+1 : GOTO 25
  24      LET M$=STR$(J) : LET MI=VAL(M$(1 TO 1))+1:
 LET MD=VAL(M$(2 TO 2))+1
  25      FOR N=1 TO 3
  26         PRINT AT N,XD;A$(MI,N);" ";A$(MD,N)
  27         NEXT N
  30      FOR K=S1 TO 59
  50         PRINT AT YS,XS;K;" "
  60         FOR L=1 TO 100
  70            NEXT L
  80         NEXT K
  85      LET S1=0
  90      NEXT J
  95   LET M1=0
 100   NEXT I
 110 LET H1=0
 120 GOTO 10
 500 REM Inicio
 510   LET XH=1 : LET XD=10
 520   LET XS=20: LET YS=3
 530    RETURN
1000 REM Lectura de los caracteres de cada cifra
1010    FOR I=1 TO 10
1020       FOR J=1 TO 3
1030          READ G,R : LET A$(I,J)=CHR$(128+G)+CHR$(128+R)
1040          NEXT J
1050       NEXT I
1060    RETURN
2000 DATA 11,7,10,5,14,13
2010 DATA 9,10,0,10,4,14
2020 DATA 3,7,4,2,14,12
2030 DATA 3,7,4,13,12,13
2040 DATA 10,0,14,14,0,10
2050 DATA 11,3,14,12,12,13
2060 DATA 11,3,14,12,14,13
2070 DATA 3,7,0,13,0,5
2080 DATA 11,7,15,15,14,13
2090 DATA 11,7,14,13,0,5
```

16.1.11 Un reloj analógico

Veamos por último cómo construir un reloj analógico, es decir, un reloj con esfera y saetas, que se vayan moviendo con el tiempo.

Para ello utilizaremos las facilidades que ofrece el ZX-SPECTRUM para realizar gráficos. Veamos el listado del programa.

```
 10 REM ================================
 20 REM RELOJ ANALOGICO
 30 REM --------------------------------
 40    CLS
 50    LET XO=177 : LET YO=87 : REM Centro de la esfera
 60    LET R = 70 : REM Radio de la esfera
 70    LET RS= 65 : REM Longitud de la segundera
 80    LET RM= 55 : REM Longitud de la minutera
 90    LET RH= 30 : REM Longitud de la horaria
100    LET RA=  5 : REM Long. lineas de las horas
110    GOSUB 550
120    CLS
140 REM Dibujo de la esfera
150    FOR I = 1 TO 60
160       LET AN = I*PI/30
170       LET X = XO+R*SIN(AN)
180       LET Y = YO+R*COS(AN)
190       PLOT X,Y
200       IF I<>INT(I/5)*5 THEN GOTO 240
210       LET XF=RA*SIN(AN)
220       LET YF=RA*COS(AN)
230       DRAW XF,YF
240       NEXT I
250    PLOT XO,YO
300 REM Movimiento de las saetas
310    FOR I = 1 TO 60
315       FOR K = 1 TO 100 : NEXT K
320       LET A = I*PI/30 : LET AN = A
330       PLOT XO,YO : DRAW INVERSE 1, XA-XO,YA-YO
350       LET X = XO+RS*SIN(AN)
360       LET Y = YO+RS*COS(AN)
370       PLOT XO,YO : DRAW X-XO,Y-YO
380       LET XA = X : LET YA=Y
390       LET AH = AHI+A/3600
400       PLOT XO,YO : DRAW INVERSE 1, XHA-XO,YHA-YO
410       LET X = XO+RH*SIN(AH)
420       LET Y = YO+RH*COS(AH)
430       PLOT XO,YO : DRAW X-XO,Y-YO
440       LET XHA = X : LET YHA = Y
450       LET AM = AMI+A/60
460       PLOT XO,YO : DRAW INVERSE 1, XMA-XO,YMA-YO
470       LET X = XO+RM*SIN(AM)
480       LET Y = YO+RM*COS(AM)
490       PLOT XO,YO : DRAW X-XO,Y-YO
500       LET XMA = X : LET YMA = Y
510       NEXT I
520    LET AMI = AM : LET AHI = AH
530    GOTO 300
550 REM Entrada de la hora
560    INPUT AT 10,5 ; "Hora: ";H
570    IF H<0 OR H>23 THEN GOTO 560
580    INPUT AT 10,5 ; "Minutos: ";M
590    IF M<0 OR M>59 THEN GOTO 580
600 REM Calculo de la posicion inicial de las saetas
610    LET AHI=PI*((H+M/60)/6)
620    LET AMI=PI*(M/30)
630    LET XA = XO+RS*SIN(PI)
640    LET YA = YO+RS*COS(PI)
650    LET XHA=XO+RH*SIN(AHI) : LET YHA=YO+RH*COS(AHI)
660    LET XMA=XO+RM*SIN(AMI) : LET YMA=YO+RM*COS(AMI)
670 RETURN
```

Tal como vemos en el listado, el programa consta de varias partes. En primer lugar, entre las líneas 40 y 100, se asignan los valores de los distintos parámetros del reloj, tal como se indica en los comentarios correspondientes.

A continuación, y en la subrutina 550, se introduce la hora, se comprueba que sea válida, y se calcula el ángulo que deberán formar las dos saetas –es decir la posicion que ocuparán– al poner el reloj en marcha. Así, la variable *AHI* contendrá el ángulo inicial de la horaria, y *AMI* el de la minutera. Al introducir las fórmulas para calcular los ángulos se utiliza el número PI, que el ordenador tiene almacenado internamente, con este nombre. Tenga cuidado al escribirlo, pues si se confunde y escribe una variable cuyo nombre esté compuesto por la letra P y la letra I el programa no funcionará.

Entre las líneas 140 y 250 se dibuja el reloj. Para ello se escribe un punto en la posición de cada minuto (son por tanto 60 puntos), y una línea en la posición de cada hora. Finalmente se sitúa el punto central de la esfera.

En este momento podemos ya proceder a dibujar las saetas. Para simular el movimiento se irá borrando la posición anterior y dibujando la nueva, de forma que las tres van avanzando, cada una según el ángulo que le corresponde.

La variable A contine el valor del ángulo que separa los minutos. Sabiendo éste, podremos calcular la posición de la segundera, la minutera y la horaria, tal como se indica en las correspondientes fórmulas, teniendo en cuenta además, que el avance de la saeta horaria es 3.600 veces inferior al de la segundera, y el de la minutera es 60 veces menor.

Capítulo 17

ESQUEMA DE CONTENIDO

17.1 FICHEROS

En el ZC-SPECTRUM existen dos formas de guardar datos en la memoria auxiliar. La primera de ellas utiliza la grabadora de cassette y la segunda utiliza el «microdrive».

17.1.1 Ficheros de cassette

En un cassette el ZX-SPECTRUM realiza copias de los datos de la memoria central. Estos datos deben pertenecer a un conjunto dimensionado, es decir, una lista o una tabla. Para grabarlos se utiliza una modificación de la instrucción SAVE, cuya sintaxis general es:

SAVE fichero DATA variable ()

Tal vez sea conveniente repasar el capítulo 8 del segundo tomo, antes de seguir. En él aprendimos a manejar la grabadora de cassette y a guardar programas.

La palabra «fichero» situada después de SAVE, significa que se colocará allí el nombre del fichero, ya sea en forma de texto entre comillas o mediante una variable textual. Detrás de él viene la palabra DATA que indica que almacenamos datos en lugar de programa.

Finalmente se indicará la variable que se quiere almacenar. Para señalar que se trata de una variable dimensionada se colocan unos paréntesis de apertura y cierre, pero sin escribir nada dentro.

Tecleemos ahora el siguiente programa:

```
10 REM Almacenamiento datos
20   DIM A(20)
30   FOR I=1 TO 20
40      LET A(I)=I*I
50      NEXT I
60   SAVE "CUADRADOS" DATA A()
70   PRINT "FIN GRABACION"
```

Este programa genera una lista de los cuadrados de los 20 primeros números naturales. La instrucción 60 almacena el conjunto A en un fichero denominado CUADRADOS. Como vemos, esta orden es parecida a la SAVE SCREEN$ estudiada en el capítulo 15 del tomo anterior.

Cuando se haya terminado la ejecución, en el cassette tendremos una copia de la lista numérica A. En realidad, CUADRADOS no es un auténtico fichero sino que simplemente es una copia de la memoria central. La información no está organizada en registros y además los datos deben ser homogéneos (en este caso todos numéricos).

Al igual que para los programas, es muy importante verificar que la información se ha grabado correctamente. Para ello utilizaremos la orden VERIFY cuya sintaxis es:

VERIFY fichero DATA variable ()

Para verificar el fichero del ejemplo, rebobinaremos el cassette y escribiremos:

VERIFY «CUADRADOS» DATA A ()

Si la verificación falla, el ordenador nos dará el siguiente mensaje:

A Tape loading error

que significa que se ha producido un error en la carga desde la cinta.

Para recuperar los datos y utilizarlos en otro programa se empleará la instrucción LOAD, cuya sintaxis general es:

LOAD fichero DATA variable ()

Para comprobar su funcionamiento utilice el comando NEW para borrar la memoria del ordenador. Teclee a continuación el siguiente programa:

```
10 REM Recuperacion datos
20   DIM A(20)
30   LOAD "CUADRADOS" DATA A()
40   FOR I=1 TO 20
50      PRINT I,A(I)
60      NEXT I
```

Pruebe el programa, pulsando RUN y a continuación la tecla PLAY de la grabadora, una vez rebobinada la cinta.

Este programa leerá la lista A de la grabadora y la traerá a memoria. A continuación las líneas 40, 50 y 60 escriben los datos en pantalla. La instrucción LOAD elimina cualquier valor previo que tuviera el conjunto A. De hecho, la línea 20 es innecesaria ya que LOAD realiza también el dimensionado. Por otra parte, no hace falta que el nombre del conjunto sea el mismo en los dos programas. En este segundo programa, la variable se podría haber denominado B o cualquier otro nombre. No olvidemos, sin embargo, que en el ZX-SPECTRUM los conjuntos dimensionados tienen el nombre limitado a una sola letra. Lógicamente, el nombre del fichero debe coincidir en ambos programas.

Hasta aquí hemos visto un fichero con datos numéricos. Para los datos textuales, es también válido lo que acabamos de explicar. En este último caso, cuando efectúe un LOAD, el ordenador le dará el mensaje «string array:» seguido del nombre. Esta frase significa conjunto de caracteres o conjunto de textos.

17.1.2. Utilización práctica

Si usted construye programas que utilizan ficheros y únicamente dispone de una grabadora de cassette, se encontrará con algunas dificul-

tades. A continuación le daremos las posibles soluciones a estos problemas.

En primer lugar, abandone los procedimientos habituales de diseño de ficheros. Ya hemos explicado que el ZX-SPECTRUM no crea ficheros auténticos en la grabadora. Se trata simplemente de copias de la memoria central. Por tanto, cuando diseñe un fichero de este tipo, tenga presente los siguientes puntos:

Punto 1. – El fichero completo debe caber en la memoria del ordenador.

Punto 2. – Los datos deben ser homogéneos, o todos numéricos o todos textuales.

Punto 3. – La forma de operar con estos ficheros también será distinta de la habitual. Al empezar la sesión de trabajo, trasladaremos el fichero a la memoria. Todas las consultas y actualizaciones se realizarán en memoria. Al finalizar la sesión de trabajo, se transferirá de nuevo el fichero completo al cassette.

El punto 1 implica una limitación importante. El tamaño de la memoria central se convierte también en el tamaño máximo de la memoria auxiliar. Es fácil que los datos que pretendemos almacenar superen en conjunto a la memoria de que disponemos. Las posibles soluciones son tres:

a) Compactar y codificar los datos. Este es un tema que veremos en otro apartado de esa lección.

b) Aumentar hasta el máximo la capacidad de nuestro ordenador mediante una expansión de memoria RAM. En el ZX-SPECTRUM, se puede llegar hasta 48 Kby disponibles.

c) Utilizar ficheros auténticos, para lo cual necesitaremos un microdrive.

La homogeneidad de los datos (punto dos) es un tema importante. Supongamos que pretendemos crear un fichero de artículos de un almacén. Para cada artículo guardaremos su descripción, la cantidad almacenada y su precio de venta. El primer dato es de tipo textual y los dos últimos son de tipo numérico. La función STR$ para convertir números en los correspondientes dígitos nos ayudará a resolver el problema. De esta forma, utilizaremos la tabla de tipo alfanumérico en la cual los valores numéricos se guardan en forma de texto. Para utilizar los números en las operaciones aritméticas, realizaremos la conversión inversa mediante la función VAL. Sin embargo, este método ocasiona un gran desperdicio de memoria, y será necesario introducir algunas variantes.

El método de trabajo del punto tres comporta el grave peligro de la pérdida de información, tanto la que se encuentra en la memoria auxiliar como la que se encuentra en la memoria central. Si después de actualizar los datos de un fichero los grabamos sobre la misma zona del cassette donde teníamos grabados los datos iniciales, nos exponemos a que se produzca una grabación defectuosa y perdamos los datos que había, al haber grabado encima de ellos, así como los datos actualizados, ya que la grabación ha sido defectuosa.

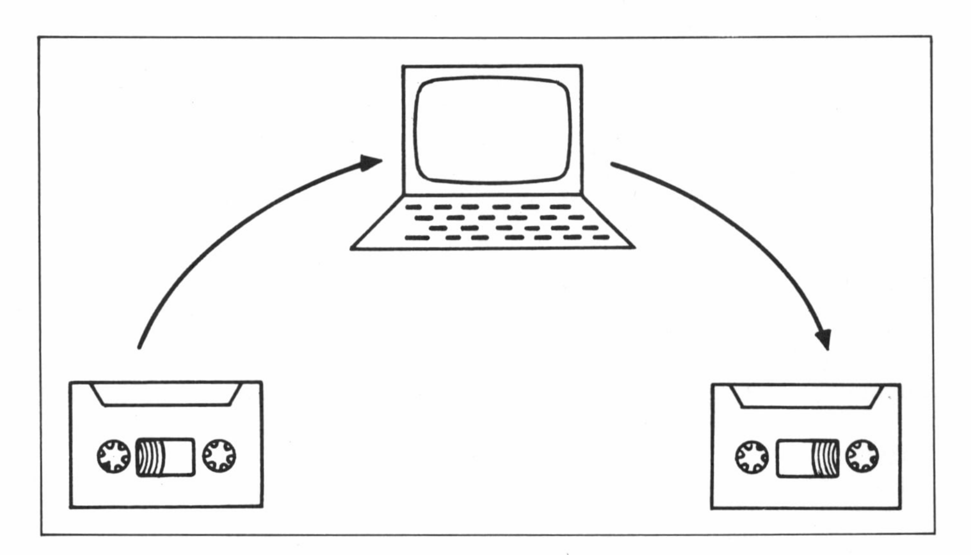


Figura 1. Proceso de actualización de un fichero de cassette.

Ciertamente, disponemos de la instrucción VERIFY para comprobarlo, pero es mejor no arriesgarse, y operar como se ve en la figura 1. Leemos los datos de un cassette y grabamos los actualizados en otro cassette o bien en la otra cara del mismo cassette. En la próxima actualización operaremos de forma inversa tomando los datos más recientes y grabándolos sobre la copia vieja. De esta manera mantenemos siempre una copia de seguridad que nos permitirá reconstruir los datos en caso de problemas. Si los datos son muy valiosos es conveniente realizar más copias en otros cassettes y guardarlos en lugar seguro.

Las perdidas de información en la memoria central se pueden producir a causa de un fallo de la corriente de alimentación. Todas las modificaciones que se hayan introducido desde el inicio de la sesión de trabajo se perderán. Contra este problema no hay solución a menos que disponga usted de un sistema de alimentación ininterrumpida.

17.1.3 Compactación y codificación

Los procedimientos de compactación y codificación son muy utilizados en informática. La necesidad de almacenar la mayor cantidad posible de información en el menor volumen posible es general en todos los equipos informáticos. Aprenderemos estas técnicas analizando un caso práctico. Reconsideremos de nuevo el fichero de artículos y ampliémoslo para que se parezca más a un fichero de un caso real. El registro o fichero de cada artículo tendría los siguientes campos:

1	Descripción	20 By
2	Proveedor	20 By
3	Localización	5 By
4	Cantidad	5 By
5	Precio coste	5 By
6	Precio venta	5 By
		60 By

El campo denominado *Localización* (3) contendrá información acerca de dónde se encuentra almacenado el producto (estantes del almacén). Como ya hemos mencionado anteriormente, los campos numéricos (los tres últimos) se utilizarán en forma textual. Supongamos que tenemos unos 100 artículos. Definiremos una tabla de 100 filas y 6 columnas. Podríamos pensar que el gasto total de la memoria es de 100 × 60 = 6.000 By, pero no es así. Como ya estudiamos en el capítulo 11 del tomo tercero, el ZX-SPECTRUM considera del mismo tamaño todos los elementos de una tabla textual. Por tanto, todos deberán tener el tamaño del mayor, es decir, 20 By. Esto hace que cada artículo consuma 6 × 20 = 120 bytes. El tamaño total del fichero alcanza ya 12 Kby.

Para reducir el tamaño del fichero deberemos pues realizar una compactación. Definiremos una tabla de solamente 3 columnas de 20 bytes, es decir:

```
DIM A$(100,3,20)
```

Entonces, los campos 3, 4, 5 y 6 se compactan para formar uno sólo. La tercera columna reparte los 20 bytes en 5 grupos de 4 bytes cada uno, tal como se muestra en la figura 2. Para extraer o introducir los grupos de 4 bytes dentro del conjunto de 20, aprovechamos el hecho de que se trata de un dato textual y por tanto podemos utilizar el operador de fragmentación. Veamos estos conceptos, creando un programa que grabe los datos iniciales de este fichero.

```
 10 REM CARGA FICHERO
 20   DIM A$(100,3,20)
 30   LET N=0
 40   INPUT "DESCRIPCION: ";D$
 50   IF D$="FIN" THEN GOTO 180
 60        LET N=N+1
 70        LET A$(N,1)=D$
 80        INPUT "Pr: ";A$(N,2)
 90        INPUT "Lo: ";L$
100        INPUT "Ca: ";C
110        INPUT "PC: ";PC
120        INPUT "PV: ";PV
130        LET A$(N,3)(1 TO 5)=L$
140        LET A$(N,3)(6 TO 10)=STR$(C)
150        LET A$(N,3)(11 TO 15)=STR$(PC)
160        LET A$(N,3)(16 TO 20)=STR$(PV)
170        GOTO 40
180        SAVE "ART" DATA A$()
```

Este programa nos va pidiendo datos y los va alamacenando en la tabla. Para la localización podemos escribir datos del tipo B-3 que signifi-

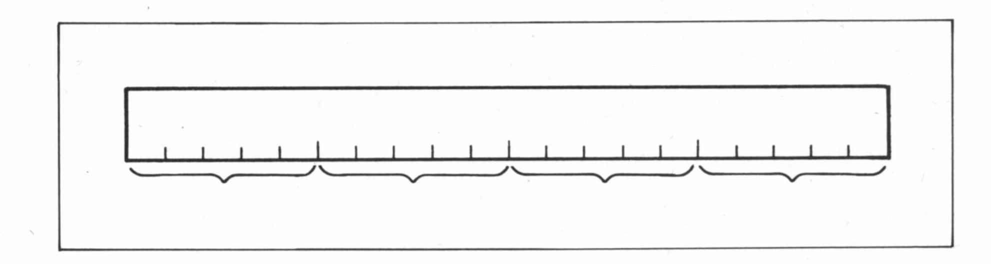


Figura 2. Compactación de cuatro campos en uno.

caría estantería B estante 3, o bien cualquier otro mientras no supere los 5 caracteres. El programa nos irá pidiendo datos hasta que tecleemos la palabra FIN (en mayúsculas) como respuesta a la descripción, y entonces los grabará en la cinta.

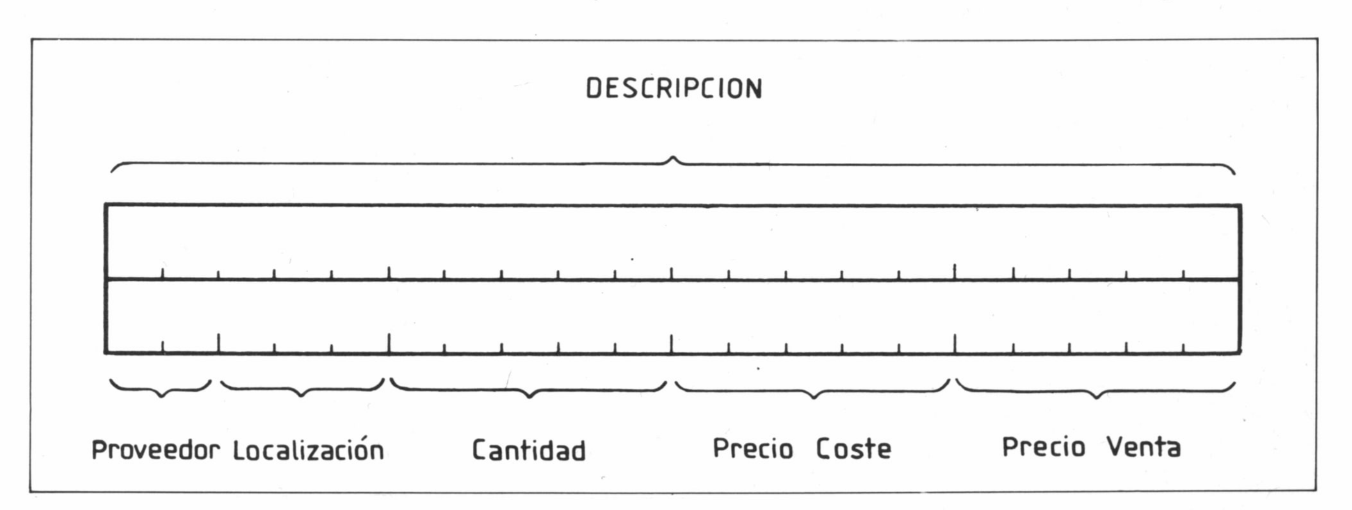
En las líneas comprendidas entre la 130 y la 160 se realiza la conversión numérico a textual y la compactación en la tercera columna de la tabla A$.

Gracias a esta compactación solamente gastamos 3 x 20 = 60 bytes por artículo. Pero, aunque no lo parezca, aún se puede reducir utilizando las técnicas de codificación.

Como sabemos, un código no es más que una tabla de traducción entre dos tipos de representación de la misma información. Por ejemplo, el código ASCII nos da la equivalencia entre la representación interna de un carácter y la representación externa legible por nosotros. De hecho, las técnicas de codificación ya se has visto en lecciones anteriores, especialmente en el capítulo 4 del primer tomo.

Los dos campos que se pueden codificar son Proveedor y Localización. Veamos cómo hacerlo. En la práctica, el número de proveedores suele ser limitado; probablemente menos de 100. Por tanto, a cada uno de ellos le podemos asignar un número de 2 cifras. De este modo, en lugar de almacenar el nombre completo, almacenaremos únicamente 2 cifras. Una técnica semejante podemos seguir para la localización. Si suprimimos el guión (innecesario ya que no aporta información alguna) tenemos un margen de aplicación desde A00 hasta Z99 que seguramente bastará para nuestros fines. De este modo, podemos hacer una compactación aún mayor de los datos, tal como se muestra en la figura 3.

Figura 3. Compactación de los datos de un registro utilizando la codificación.



En este caso, la tabla de datos se definirá así:

```
DIM A$(100,2,20)
```

con lo cual el gasto queda reducido a 2 × 20 = 40 Bytes por artículo.

Lógicamente, aparte habrá que guardar una tabla con los nombres de los proveedores.

La nueva versión del programa anterior es (aproveche las líneas que van de la 10 a la 120)

```
 10 REM CARGA FICHERO
 20   DIM A$(100,2,20)
 30   LET N=0
 40   INPUT "DESCRIPCION: ";D$
 50   IF D$="FIN" THEN GOTO 190
 60        LET N=N+1
 70        LET A$(N,1)=D$
 80        INPUT "Cod: ";P$
 90        INPUT "Lo: ";L$
100        INPUT "Ca: ";C
110        INPUT "PC: ";PC
120        INPUT "PV: ";PV
130        LET A$(N,2)(1 TO 2)=P$
140        LET A$(N,2)(3 TO 5)=L$
150        LET A$(N,2)(6 TO 10)=STR$(C)
160        LET A$(N,2)(11 TO 15)=STR$(PC)
170        LET A$(N,2)(16 TO 20)=STR$(PV)
180        GOTO 40
190        LET A$(N+1,1)="*"
200        SAVE "ART2" DATA A$()
```

Este programa crea un fichero denominado ART2. Su funcionamiento es idéntico al anterior, excepto que en el caso del proveedor se contestará su código (un número de 1 a 20) y la localización se escribirá con un máximo de 3 letras o dígitos.

Aunque obviamente se puede entrar cualquier dato, utilice los siguientes para realizar las pruebas iniciales:

D	P	L	C	PC	PV
JABÓN,	2,	A1,	200,	90	100
PERFUME,	1,	B3,	370,	500,	650
DETERGENTE 5,	3,	A2,	160,	600,	780
DETERGENTE 1,	3,	A2,	500,	200,	250
COLONIA,	5,	B1,	450,	600,	800

Después de entrar el último artículo (COLONIA), escriba FIN cuando el programa pregunte de nuevo la descripción. En este momento se grabarán los datos, contenidos en la tabla A$. La línea 190 sirve para colocar una marca de final a la lista de nombre.

De momento hemos realizado un programa para efectuar la carga inicial de los datos.

Veamos seguidamente cómo se construiría un programa que utilizara estos datos. Por ejemplo, para realizar una valoración de los artículos almacenados. El listado sería el siguiente:

```
 10 REM VALORACION ALMACEN
 20   LOAD "ART2" DATA A$()
 30   LET TC=0 : LET TV=0
 40   FOR I=1 TO 100
 50        IF A$(I,1,1)="*" THEN GOTO 130
 60        LET C=VAL(A$(I,2)(6 TO 10))
 70        LET PC=VAL(A$(I,2)(11 TO 15))
 80        LET PV=VAL(A$(I,2)(16 TO 20))
 90        LET TC=TC+C*PC
100        LET TV=TV+C*PV
110        PRINT A$(I,1),C
120        NEXT I
130   PRINT "VALORACION P. COSTE= ";TC
140   PRINT "VALORACION P. VENTA= ";TV
```

La línea 20 realiza una carga en memoria del fichero ART2. Por tanto, será necesario tener la grabadora preparada con el cassette rebobinado. Una vez se han transferido los datos sobre el conjunto A$ se opera de forma normal. En este caso se calcula el valor total de los artículos almacenados según el precio de coste y según el precio de venta. Puesto que los datos estaban compactados, se realiza una operación previa de descompactación antes de efectuar los cálculos. Por otra parte, los datos están codificados. Puesto que sólo utilizamos una parte de la información contenida en el fichero, la decodificación es muy sencilla y consiste simplemente en transformar las tres cantidades que están en forma textual, a su correspondiente forma numérica.

El programa escribe una línea para cada artículo considerado. Si los datos se han almacenado correctamente, el resultado que debe aparecer en pantalla es:

```
JABÓN           200
PERFUME         370
DETERGENTE 5    160
DETERGENTE 1    500
COLONIA         450
VALORACIÓN P. COSTE = 669000
VALORACIÓN P. VENTA = 870300
```

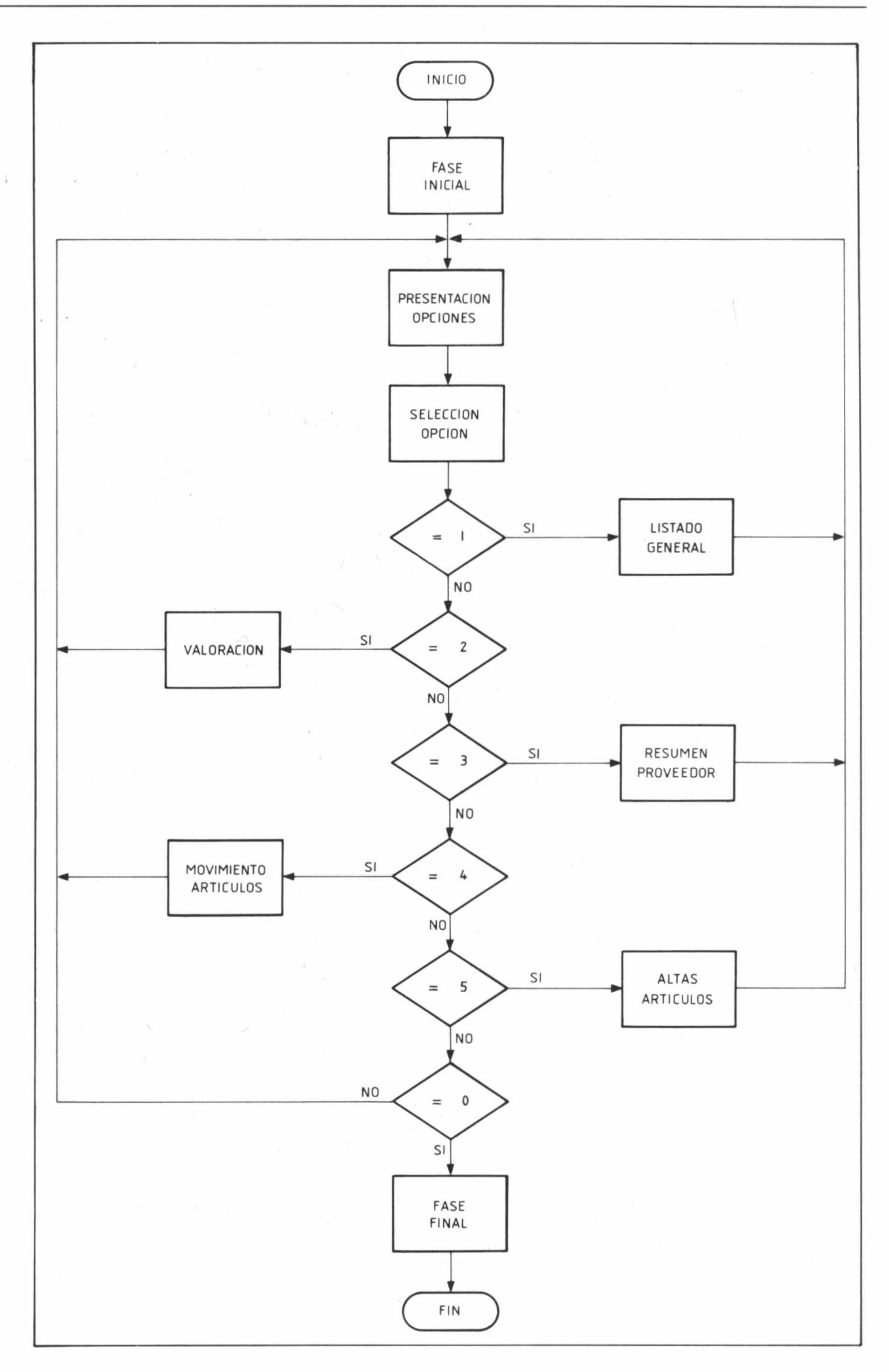


Figura 4. Ordinograma para el control de un almacén de artículos.

17.1.4 Caso práctico: Control almacén

A continuación veremos un programa completo para el control del almacén. Este programa incluye todas aquellas opciones mínimas que se

necesitan para llevar el control, aunque se podrían añadir algunas ampliaciones, según las necesidades concretas de cada usuario.

Aunque este programa está diseñado para un proceso concreto, incluye todas las operaciones típicas que se emplean en la gestión de ficheros. Si se entiende bien este programa, efectuando unos retoques podremos aprovecharlo para cosas tan dispares como llevar un control de clientes, una agenda personal, o un control de producción.

Ciertamente, en estos casos habrá que cambiar las instrucciones, pero el diseño básico seguirá siendo el mismo. En la figura 4 se muestra un diagrama del diseño. Como se puede apreciar, se ha seguido una técnica modular, de modo que el programa se pueda construir pieza a pieza. Los bloques que constituirán el programa son:

1. Fase inicial: Se efectúa una sola vez cuando se pone en marcha el programa. Se leen los datos del fichero. La subrutina empezará en la línea 7.000.

2. Presentación Menú y selección de una opción: Estas operaciones estarán en el programa principal.

3. Listado general. Se escribirán los datos tal como están en memoria. La subrutina empieza en la línea 1.000.

4. Valoración. Se calcula el valor de lo que está almacenado según el precio de coste y según el precio de venta. Coincide con el programa anterior. La subrutina empieza en la línea 2.000.

5. Resumen proveedores. Se escribe un listado con el total suministrado por cada proveedor. La subrutina correspondiente empieza en la línea 3.000.

6. Movimiento artículos. Se registran las entradas o salidas de un artículo a fin de mantener actualizado el fichero. La subrutina empieza en la línea 4.000.

7. Altas artículos. Este módulo se emplea cuando adquirimos un artículo nuevo que no consta en el almacén. La subrutina empieza en la línea 5.000.

8. Fase final. Cuando el programa termine se pasará control al módulo final, que transfiere los datos de nuevo a la grabadora. Este módulo empieza en la línea 6.000.

El listado del programa principal es el siguiente:

```
10 REM Fichero articulos
20   GOSUB 7000
30   CLS
40   PRINT TAB(10);"CONTROL ALMACEN"
50   PRINT : PRINT TAB(5);"0.- FIN"
60   PRINT TAB(5);"1.- LISTADO GENERAL"
70   PRINT TAB(5);"2.- VALORACION ALMACEN"
80   PRINT TAB(5);"3.- RESUMEN PROVEEDORES"
90   PRINT TAB(5);"4.- MOVIMIENTOS ARTICULOS"
```

```
100   PRINT TAB(5);"5.- ALTAS PRODUCTOS"
110   INPUT "OPCION: ";OP
120   IF OP=1 THEN GOSUB 1000 : GOTO 30
130   IF OP=2 THEN GOSUB 2000 : GOTO 30
140   IF OP=3 THEN GOSUB 3000 : GOTO 30
150   IF OP=4 THEN GOSUB 4000 : GOTO 30
160   IF OP=5 THEN GOSUB 5000 : GOTO 30
170   IF OP=0 THEN GOSUB 6000
180   GOTO 110
```

Como se puede apreciar facilmente, el programa consta de tres partes. La primera de ellas es la inicialización, que se efectúa en una subrutina (línea 20). La segunda es la presentación en pantalla de las opciones, llamada menú (líneas 30 a 100). Finalmente se selecciona una opción pasando control a la subrutina adecuada (líneas 110 a 180).

Una vez entrado el programa principal, seguiremos con la subrutina de inicialización. El listado es el siguiente:

```
7000 REM Inicial
7010      LOAD "ART2" DATA A$()
7020      FOR N=1 TO 100
7030         IF A$(N,1,1)="*" THEN GOTO 7050
7040         NEXT N
7050      LET N=N-1
7060      DIM P$(20,20) : LET NP=5
7070      FOR I=1 TO NP
7080         READ P$(I)
7090         NEXT I
7100      DIM T(20)
7110      DATA "PERFUMES SOL, S.A."
7120      DATA "JABONES DOMESTICOS"
7130      DATA "IND. DETERGENTE"
7140      DATA "SUM. PAPELERIA, S.A."
7150      DATA "DISTIBUIDORA GRAL."
7160      RETURN
```

La primera operación que se realiza es la carga del fichero (línea 7.010). A continuación, en las líneas 7.020 a 7.050 se determina el número de artículos mediante la búsqueda de la marca de final. Por último se establecen los datos de los proveedores. En la línea 7.060 se define una tabla de 20 proveedores con 20 letras para el nombre. Estos nombres se obtienen de las instrucciones DATA. Por el momento sólo se han definido 5 nombres de los 20 posibles (variable NP de la línea 7.070), pero se puede modificar fácilmente para ampliar la lista de proveedores.

Aprovechando el diseño modular del programa, las subrutinas siguientes las iremos añadiendo y probando una a una. La primera de ellas es la que realiza el listado general de artículos. El problema principal es

que la anchura de la pantalla es muy reducida. Por consiguiente, la información de cada artículo ocupará tres líneas. El listado es:

```
1000 REM Listado
1010  CLS
1020  FOR I=1 TO N
1030         PRINT I;TAB(4);A$(I,1);" ";A$(I,2)(2 TO 5)
1040         LET R$=A$(I,2)(1 TO 2)
1050         PRINT TAB(4);P$(VAL(R$));"C=";A$(I,2)(6 TO 10)
1060         PRINT TAB(4);"PC=";A$(I,2)(11 TO 15);
1070         PRINT TAB(20);"PV=";A$(I,2)(16 TO 20)
1080         NEXT I
1090      PAUSE 0
1100      RETURN
```

La estructura de esta subrutina es muy sencilla y se limita a escribir la información de cada artículo. En la primera línea de la pantalla se escribirá el número de artículo, la descripción y la localización. En la segunda se escribirá el nombre del proveedor, que se obtiene decodificando el número almacenado (instrucciones 1.040 y 1.050), y la cantidad. En la tercera línea se escribirán los precios de coste y de venta.

Una vez llegados a este punto ya podemos probar el trozo de programa que hemos escrito. Recuerde que aprovechamos el fichero ART2 construido con el programa anterior. Al ejecutar ahora este programa, se cargan en memoria los datos del fichero. Seguidamente aparece en pantalla el menú de opciones. De momento sólo podemos probar la opción 1. Al pedirla, aparecerá en pantalla un lista de la forma siguiente:

```
1 JABÓN              A1
  JABONES DOMÉSTICOS   C = 200
  PC = 90            PV = 100

2 PERFUME            B3
  PERFUMES SOL         C = 370
  PC = 500          PV = 650

  .
  .
  .
```

Al final, el programa se espera a que pulsemos una tecla cualquiera y aparece de nuevo en pantalla el menú de opciones. Si todo ha funcionado correctamente, ya puede interrumpir el programa y añadir nuevas subrutinas.

La siguiente opción ya la hemos visto anteriormente y consiste en efectuar una valoración de los artículos. El listado de la subrutina es:

```
2000 REM Valoracion almacen
2010 CLS
2020 LET TC=0 : LET TV=0
2030 FOR I=1 TO N
```

```
2040         LET C=VAL(A$(I,2)(6 TO 10))
2050         LET PC=VAL(A$(I,2)(11 TO 15))
2060         LET PV=VAL(A$(I,2)(16 TO 20))
2070         LET TC=TC+C*PC
2080         LET TV=TV+C*PV
2090         NEXT I
2100 PRINT "VALORACION P. COSTE=";TC
2110 PRINT "VALORACION P. VENTA=";TV
2120 PAUSE 0
2130 RETURN
```

El listado no requiere apenas explicación, puesto que coincide con el programa visto anteriormente. Únicamente hacer notar la instrucción PAUSE de la línea 2.120 para asegurar que el resultado permanezca en pantalla hasta que el usuario decida seguir.

La tercera opción nos permitirá obtener un resumen de lo suministrado por cada proveedor. El listado es el siguiente:

```
3000 REM Resumen proveedor
3010      CLS
3020      FOR I=1 TO 20 : LET T(I)=0 : NEXT I
3030      FOR I=1 TO N
3040           LET P=VAL(A$(I,2)(1 TO 2))
3050           LET C=VAL(A$(I,2)(6 TO 10))
3060           LET PC=VAL(A$(I,2)(11 TO 15))
3070           LET T(P)=T(P)+C*PC
3080           NEXT I
3090      PRINT "PROVEEDOR";TAB(20);"VALOR"
3100      PRINT "-------------------------------"
3110      LET A=0
3120      FOR I=1 TO NP
3130           PRINT P$(I);T(I)
3140           LET A=A+T(I)
3150           NEXT I
3160      PRINT "TOTAL";TAB(20);A
3170      PAUSE 0
3180      RETURN
```

La primera opción es la puesta a cero de los totales (línea 3.020). A continuación, en el bucle que va de la 3.030 hasta la 3.080 se acumula el valor de cada artículo a precio de costa para el proveedor correspondiente. La última parte de la subrutina escribe el nombre y el total de cada proveedor.

Naturalmente, cada vez que se añada una subrutina, es conveniente que realice algunas pruebas para asegurar el correcto funcionamiento. Las opciones que hemos visto hasta el momento extraían la información del fichero. Veamos ahora las opciones que añaden información, o alteran la existente.

La subrutina 4.000 permite actualizar la existencia de un artículo en el almacén. El listado es:

```
4000 REM Movimiento articulos
4010         CLS : PRINT "MOVIMIENTO ARTICULOS"
4020         INPUT "NUMERO ARTICULO:";I
4030         PRINT AT 2,1 ;A$(I,1)
4040         INPUT "CONFORME (S/N):";R$
4050         IF R$<>"S" THEN GOTO 4010
4060         INPUT "ENTRADA O SALIDA (E/S):";R$
4070         INPUT "CANTIDAD:";Q
4080         LET C=VAL(A$(I,2)(6 TO 10))
4090         IF R$="S" THEN GOTO 4140
4100         INPUT "PRECIO COMPRA";PC
4110         LET A$(I,2) (11 TO 15)=STR$(PC)
4120         LET C=C+Q
4130         GOTO 4170
4140         INPUT "PRECIO VENTA:";PV
4150         LET A$(I,2) (16 TO 20)=STR$(PV)
4160         LET C=C-Q
4170         LET A$(I,2)(6 TO 10)=STR$(C)
4180         RETURN
```

Inicialmente, la subrutina nos pide el número del artículo (el que aparece en el listado general). Una vez contestado el número, presenta en pantalla la descripción del artículo y nos pide que confirmemos que no nos hemos equivocado de número. Seguidamente pide si se trata de un movimiento de entrada (adquisición) o de salida (venta), así como la cantidad.

Si se trata de una entrada, nos pregunta el precio de compra, y si se trata de una salida nos pide el precio de venta. En ambos casos, el precio pasa a ser el valor almacenado en el fichero.

En el caso de salida, se descuenta la cantidad y en caso de entrada se suma la cantidad a la existencia inicial.

Observe que todas las modificaciones se han realizado en memoria y solamente al finalizar el programa se transferirán al fichero magnético. Igual procedimiento se sigue con la operación de dar de alta. El listado es:

```
5000 REM Altas
5010         CLS : PRINT "ALTAS ARTICULOS"
5020         INPUT "DESCRIPCION:";D$
5030         IF LEN(D$)=0 THEN RETURN
5040         LET N=N+1 : IF N=100 THEN RETURN
5050         LET A$(N,1)=D$
5060         PRINT "NUMERO=";N
5070         INPUT "COD. PROVEEDOR";R$
5080         LET A$(N,2)(1 TO 2)=R$
5090         PRINT "PROVEEDOR:";P$(VAL(R$))
5100         INPUT "LOCALIZACION: ";R$
5110         LET A$(N,2)(3 TO 5)=R$
5120         INPUT "CANTIDAD:";C
5130         INPUT "P. COMPRA:";PC
```

```
5140        INPUT "P. VENTA:";PV
5150        LET A$(N,2)(6 TO 10)=STR$(C)
5160        LET A$(N,2)(11 TO 15)=STR$(PC)
5170        LET A$(N,2)(16 TO 20)=STR$(PV)
5180        LET A$(N+1,1)="*"
5190        RETURN
```

En la línea 5.020 pregunta la descripción. En la siguiente, comprueba que el texto no sea vacío y en caso contrario retorna al programa principal. Esta línea se ha colocado para preveer posibles errores. En efecto, si por equivocación pide las opciones 1, 2 o 3 no ocurre nada especial y simplemente bastará esperar a que termine el listado. Si la opción elegida erróneamente es la 4, bastará con responder una cantidad nula para que el fichero no se vea afectado. Sin embargo, supongamos que inadvertidamente pide la opción 5 cuando en realidad quería otra. Si no existiese la línea 5.030 se vería obligado a añadir un artículo al fichero, aunque fuera un artículo imaginario o falso.

Recuerde que no se puede interrumpir el programa, pues las modificaciones se almacenan en memoria y se perderían. Gracias a la línea 5.030, basta con que responda el texto vacío para que quede anulada la opción 5 y el programa vuelve al menú inicial.

Las instrucciones siguientes de esta subrutina piden el resto de datos del artículo. En pantalla se presenta el número asignado al nuevo artículo. En la línea 5.170 la marca de final –un asterisco (*)– se corre una posición hacia adelante.

Una vez haya comprobado exahustivamente el funcionamiento de las opciones, se deberá añadir la subrutina de finalización. El listado es muy sencillo:

```
6000 REM Fin
6010        INPUT "CONFIRMA (S/N):";R$
6020        IF R$<>"S" THEN RETURN
6030        SAVE "ART2" DATA A$()
6040        STOP
```

En esta subrutina, después de pedir la confirmación, se graban los datos en la cinta. Puesto que aquí finaliza el proceso, la subrutina termina con una instrucción STOP.

Tenga en cuenta las normas de grabar a la hora de hacerlo.

En este momento ya tenemos construido un programa para controlar el almacén. Ciertamente se podrían añadir nuevas opciones, como la posibilidad de cambiar el proveedor de un artículo o el dar de baja un artículo. También sería importante añadir las validaciones de los datos de entrada para impedir que se introduzcan datos erróneos. Una ampliación fácil de realizar consistiría en aprovechar las subrutinas vistas en el capítulo 13 del tomo anterior para encolumnar números. De este modo los resultados aparecerían mejor presentados.

Este programa, aunque ha sido diseñado y construido para un caso

concreto, puede tomarlo como modelo para construir programas que manejan ficheros con una grabadora de cassette. En concreto, debe recordar los siguientes puntos:

- El mecanismo es: a) Carga en memoria, b) Actualización en memoria y c) Transferencia a la grabadora al finalizar.
- En pantalla se presentará un menú de opciones.
- Una parte de las opciones consistirá en listados y resúmenes de los datos almacenados.
- La otra parte serán las opciones de dar de alta, dar de baja y modificar o actualizar un registro ya existente.
- Las opciones delicadas deben solicitar confirmación antes de seguir.

17.1.5 Ficheros con microdrive

Este apartado está dedicado principalmente a aquellos usuarios que dispongan de una unidad de microdrive. Sin embargo, es conveniente leer esta parte aunque usted no vaya a utilizarla, puesto que hay algunos conceptos y programas de uso general que le permitirán conocer con más profundidad a su ordenador. Para facilitarle la tarea, los programas llevan un REM en donde se indica si para ejecutarlos es necesario disponer de microdrive, impresora u otra dispositivo. Los comandos o las instrucciones ejecutadas en modo inmediato (sin número de línea), no llevan indicación, pero los requerimientos se deducen fácilmente del texto.

Si su ZX-SPECTRUM dispone de una (o más) unidad de microdrive; podrá manejar auténticos ficheros con bastante rapidez. Para conectar el microdrive debe disponerse de la ZX INTERFACE 1.

La unidad de microdrive maneja unos cartuchos intercambiables con una capacidad de al menos 80 Kby cada uno. Se pueden conectar hasta 8 unidades de microdrive simultáneamente, con lo que la capacidad total sería de al menos 640 Kby. Antes de utilizar un cartucho nuevo, debe inicializarse o formatearse.

El cartucho es, en realidad, un pequeño cassette de construcción especial. Por consiguiente, los ficheros serán obligatoriamente de tipo secuencial ya que el dispositivo por su propia naturaleza es secuencial. No obstante, dada la gran rapidez con que opera el microdrive, el ZX-SPECTRUM ofrece algunas ampliaciones al funcionamiento típicamente secuencial.

Sobre un cartucho pueden realizarse exactamente las mismas operaciones que sobre un cassette, explicadas en el capítulo 8.

Para grabar programas en el microdrive se utiliza una versión modificada del comando SAVE. La sintaxis es:

SAVE *«m»;1;«nombre»

El asterisco (*) y la letra m son fijos. El número 1 indica que nos referimos a la primera unidad de microdrive (puede haber hasta 8, tal

como hemos dicho). El «nombre» puede ser cualquier texto de un máximo de 10 símbolos. Por ejemplo, pruebe el siguiente programa que escribe un triángulo de asteriscos.

```
10 REM TRIANGULO
20 REM USO GENERAL
30   FOR I=0 TO 10
40       PRINT TAB(15-I);
50       FOR J=1 TO 2*I+1
60           PRINT "*";
70           NEXT J
80       PRINT
90       NEXT I
```

Para grabarlo en el microdrive, escriba:

SAVE *«m»;1;«triángulo»

Como puede comprobar, la puesta en marcha y detención del microdrive es automática. Para verificar que la grabación ha sido correcta, escriba:

VERIFY *«m»;1;«triángulo»

Aunque se trata de un dispositivo secuencial, no hace falta preocuparse de rebobinar el cartucho. El microdive se encarga de todo y transfiere el programa a la memoria. Al igual que en la grabadora de cassette, los programas pueden almacenarse preparados para la ejecución inmediata. Por ejemplo, grabe de nuevo el programa anterior haciendo:

SAVE *«m»;1;«triángulo2» LINE 10

Borre la memoria con el comando NEW y escriba:

LOAD *«m»;1;«triángulo2»

Una vez cargado, el programa se ejecutará automáticamente empezando por la línea 10 sin necesidad de utilizar el comandio RUN.

Modificando el comando MERGE, podrá añadir un programa que esté almacenado en el microdrive a las instrucciones que tenga en memoria. Escriba NEW y teclee el siguiente segmento de programa:

```
100  FOR I=0 TO 10
110      PRINT AT I,0;
120      FOR J=0 TO 30
130          PRINT " ";
140          NEXT J
150      NEXT I
```

Este programa realiza un borrado de la zona de pantalla donde se dibuja el triángulo. Para añadir este segmento al programa anterior, escriba:

MERGE * «m»;1;«triángulo»

Ahora en memoria tiene dos programas formando uno solo. Recuerde, sin embargo, que el comando MERGE no funciona si el programa ha sido grabado con ejecución automática, es decir, con SAVE...LINE. Por esta razón, se ha utilizado el programa «triángulo» en lugar de «triángulo2». Si se aplica el comando MERGE sobre este fichero, aparece el mensaje

Merge error

El ZX-SPECTRUM le ofrece además una opción de carga y ejecución automática para aquellos programas que se utilicen repetidamente. Para conseguir este efecto, debe seguir las siguientes reglas:

a) El programa debe llamarse run.

b) El cartucho que lo contiene se introducirá en el microdrive 1.

c) La carga y ejecución automática solamente funciona después de introducir NEW. Por supuesto, también funciona cuando se acaba de poner en marcha el ordenador.

Por ejemplo, el programa anterior podría grabarse de esta forma haciendo:

SAVE *«m»;1;«run» LINE 10

La palabra run hay que escribirla letra a letra. No sirve la tecla RUN. Para ejecutarlo, escriba:

NEW
RUN

En este último caso, RUN es precisamente el comando de ejecución y debe pulsarse la tecla RUN. Operando de esta forma, se ejecuta el programa anterior. Este procedimiento es útil si se quiere dejar un programa en manos de personas inexpertas. De esta manera, bastará con que pongan en marcha el ordenador y pulsen una sola tecla (la tecla RUN) para que el programa se ejecute.

17.1.6 Acceso a un fichero

17.1.6.1 Apertura. Instrucción OPEN

En el ZX-SPECTRUM, con la instrucción OPEN se pueden abrir ficheros y dispositivos. Los ficheros se identifican por su nombre. Por otra

parte, cada dispositivo tiene una letra asignada para identificarlo. Estas letras son:

b	binary	Interfaz RS-232 con datos en binario.
k	keyboard	Teclado del ZX-SPECTRUM y zona de trabajo de pantalla.
m	microdrive	Unidad de microdrive.
n	net	Red local de ordenadores.
p	printer	Impresora ZX-SPECTRUM Printer.
s	screen	Zona de presentación de la pantalla.
t	text	Interfaz RS-232 con datos textuales.

La interfaz RS-232 es un dispositivo adicional del ZX-SPECTRUM que permite la conexión de impresoras estándar, de modems (dispositivos para la comunicación telefónica entre ordenadores) o de cualquier otro dispositivo que utilice precisamente el tipo de conexión sobre RS-232. Por esta interfaz se pueden enviar textos (dispositivo t), que estarán formados por caracteres imprimibles y quedarán descartados los códigos de control. También se puede enviar cualquier tipo de dato, para lo cual se utiliza el dispositivo b.

La red local es un conjunto de ordenadores ZX-SPECTRUM conectados entre sí, de modo que se pueden recibir y transmitir datos y programas.

Para designar a los dispositivos se pueden emplear las letras en minúsculas o mayúsculas indistintamente.

Con la instrucción OPEN se pueden abrir hasta 16 canales que se asignarán a dispositivos o ficheros. No obstante, puesto que el BASIC a su vez también necesita utilizar los dispositivos, realiza unas aperturas iniciales cuando se pone en marcha el ordenador. Los canales abiertos de entrada son:

0 Salida a la parte inferior de la pantalla (zona de trabajo).
Este canal está asignado al dispositivo K.

1 Entrada desde el teclado.
Está asignado al dispositivo K.

2 Salida a la zona de presentación de la pantalla.
Está asignado al dispositivo S.

3 Salida a la impresora. Está asignado al dispositivo P.

Por consiguiente, el usuario dispone de los canales comprendidos entre el 4 y el 15. Tenga en cuenta qué números de canal distintos pueden estar asignados al mismo dispositivo, pero no al revés. Haciendo una analogía, podríamos decir que varias tuberías pueden llevar agua al mismo depósito, pero la misma tubería no puede llevar agua a dos depósitos a la vez.

Dado que la instrucción OPEN únicamente tiene sentido si se utiliza en combinación con las instrucciones de acceso (INPUT, PRINT,...), en este apartado solamente daremos una explicación general y los ejemplos de utilización se verán en el apartado siguiente.

En el ZX-SPECTRUM, la instrucción OPEN tiene tres posibles sintaxis según sea el dispositivo que se va a usar.

– *Dispositivos K, P, y S*

La sintaxis general es

OPEN número, «letra»

La palabra OPEN se encuentra en la tecla del 4. Lleva incluido el símbolo de sostenido. Por ejemplo:

```
10   OPEN #4,"S"     El canal 4 se asigna a la pantalla
30   OPEN #10,"P"    Se abre la impresora por el canal 10
```

– *Dispositivos B, N y T:*

La sintaxis general es:

OPEN número;«letra»

Observe que se utiliza un punto y coma (;) en lugar de una coma (,) Ejemplos:

```
100  OPEN #4;"t"     El canal 4 se asigan a la interfaz
                     RS-232 para salida de texto
10   OPEN #8;"b"     El canal 8 se asigna a la interfaz
                     RS-232 para salida de datos en binario.
```

En el caso del dispositivo N hay que especificar además, el número que tiene el ordenador en la red. Por ejemplo:

```
10   OPEN #7;"n";2
```

El canal 7 se asigna al ordenador 2 de la red. Este canal puede ser de entrada o de salida según sea la primera operación que realice. Este número puede estar comprendido entre 0 y 64.

– *Dispositivo M*

Puesto que se trata de microdrive, además del dispositivo hay que indicar el nombre del fichero. La sintaxis es:

OPEN número;«m»;1;«nombre»

Esta sintaxis es para el primer microdrive. En caso contrario hay que

cambiar el 1 por el valor adecuado. Los ficheros del microdrive son siempre de tipo secuencial. El nombre del fichero puede ser cualquiera mientras no supere los 10 símbolos. Ejemplo:

```
50   OPEN #5;"m";1;"CLIENTES"
```

Esta instrucción abre el fichero CLIENTES situado en el microdrive 1 a través del canal 5. Si al realizar el OPEN, el fichero ya existe, este se abre para lectura. Por el contrario, si el fichero no existe, se abre para salida. En el microdrive coexisten los ficheros que contienen datos junto con los que contienen programas. El ZX-SPECTRUM impide que se empleen de forma errónea. Por ejemplo, si intenta realizar un OPEN sobre un fichero que contiene un programa, el ordenador le dará el mensaje

Wrong file type

indicándole que utiliza un tipo equivocado de fichero. Por último recuerde que los canales de 0 a 3 están ya abiertos.

No utilice estos números en una instrucción OPEN.

17.1.6.2 Lectura y grabación

Para acceder a un fichero se utilizan versiones modificadas de las instrucciones PRINT e INPUT. Veamos primero la forma de enviar datos a través de un canal hacia un dispositivo o hacia un fichero. La instrucción fundamental tiene la sintaxis

PRINT número; lista expresiones

Es decir, se trata de una instrucción PRINT normal añadiéndole el número de canal. Cada vez que se ejecuta esta instrucción se envía un registro al dispositivo o al fichero.

Como ya se ha indicado, el BASIC mantiene abiertos de entrada 4 canales. La instrucción PRINT normal utiliza el canal 2 (zona de presentación de la pantalla). Por tanto, las dos instrucciones siguientes son equivalentes

```
PRINT "ESCRITO EN PANTALLA"
PRINT #2;"ESCRITO EN PANTALLA"
```

Por su parte, la instrucción LPRINT utiliza el canal 3 (asignado a la impresora ZX). Sin embargo, eta asociación se puede cambiar. Pruebe

```
LPRINT #2;"ESCRITO EN PANTALLA"
```

El texto aparece de nuevo en pantalla. Pueden establecerse nuevos canales para enviar datos a un dispositivo. Por ejemplo, escriba el programa

```
10 REM USO GENERAL
20   OPEN #4,"S"
30   PRINT #4;"ESCRITO EN PANTALLA"
```

Al teclear RUN, el resultado aparecerá de nuevo en pantalla. Una interesante ampliación de esta posibilidad permite que el usuario elija la unidad de salida. Para verlo, introduzca el siguiente programa:

```
10 REM REQUIERE IMPRESORA
20   INPUT "PANTALLA O IMPRESORA (I/P): ";A$
30   LET U$="S" : IF A$="I" THEN LET U$="P"
40   OPEN #4,U$
50   FOR I=1 TO 5
60        PRINT #4;I,SQR(I)
70        NEXT I
```

En la línea 10, el usuario elije si quier obtener los datos en pantalla o en impresora. La instrucción OPEN se encarga de hacer la asignación. Observe que la instrucción PRINT de la línea 60 no varía. Siempre escribe utilizando el canal 4. Lo que pasa es que este canal conducirá los datos a la pantalla o a la impresora según la asignación realizada por la instrucción OPEN.

Si en lugar de enviar los datos a un dispositivo queremos enviarlos a un fichero, el procedimiento será análogo. Por ejemplo, escriba el siguiente programa para crear un fichero de direcciones.

```
 10 REM REQUIERE MICRODRIVE
 20   OPEN #4;"M";1;"DIRECCION"
 30   FOR I=1 TO 4
 40        READ N$,D$
 50        PRINT #4;N$,D$
 60        NEXT I
 70   CLOSE #4
 80   DATA "JUAN", "C/ MAYOR 15"
 90   DATA "ANA","PL. NUEVA 5"
100   DATA "LUIS","AV. PRINCIPAL 200"
110   DATA "MARIA","PASEO DE LA LUNA S/N"
```

La instrucción CLOSE se encuentra sobre la tecla del 5. Se explicará más adelante, pero en este programa es necesario utilizarla. Este programa crea un fichero llamado DIRECCIÓN en el microdrive 1 y lo llena con los nombres y direcciones contenidos en las instrucciones DATA. Este

fichero contendrá pues, 4 registros. Ahora que ya disponemos de un fichero, veamos cómo leerlo. Para ello utilizamos una variante de la instrucción INPUT cuya sintaxis es:

INPUT número;lista variables

Para leer los datos del fichero anterior, escriba el programa:

```
NEW
10 REM REQUIERE MICRODRIVE
20    OPEN #8;"m";1;"DIRECCION"
30    FOR I=1 TO 4
40         INPUT #8;A$
50         PRINT A$
60         NEXT I
70    CLOSE #8
```

En la instrucción OPEN asegúrese de que el nombre del fichero coincida con el del programa anterior, especialmente por lo que se refiere a mayúsculas y minúsculas. En la línea 40 se lee un registro y a continuación se escribe en pantalla. Probablemente encontrará extraño que se utilice una sola variable en lugar de las dos que utilizaba el programa anterior. Más tarde insistiremos sobre este punto. En este programa leemos 4 veces del fichero porque sabemos que contiene cuatro registros. Si se intentan leer más registros de los existentes, el ordenador da el mensaje

End of file

que indica que se ha llegado al final del fichero. Para leer los datos de un fichero también se puede emplear la función INKEY$.

Esta función obtiene cada vez un carácter del fichero. Escriba el programa

```
10 REM REQUIERE MICRODRIVE
20    OPEN #8;"m";1;"DIRECCION"
30    LET A$=INKEY$#8
40    PRINT A$;
50    GOTO 30
60    CLOSE #8
```

Este programa va escribiendo el contenido del fichero letra a letra. Al final escribirá el mensaje

End of file

En ambos programas, la instrucción OPEN encuentra el fichero en

el microdrive y por consiguiente lo abre para lectura. Cualquier intento de escritura dará el error

Writing to a «read» file

Sin embargo, a veces no es tan evidente la escritura. Por ejemplo, la instrucción

```
INPUT #8;"NOMBRE:";A$
```

incluye una escritura y por tanto daría un error. Menos evidente es la instrucción

```
INPUT #8;A$,B$
```

La coma que separa las variables de una lista, en el ZX-SPECTRUM tiene el efecto de tabulación. Esta tabulación implica una escritura y de nuevo obtendríamos un error. Por tanto, un INPUT que lea de fichero no incluirá ningún texto y las variables se separarán mediante punto y coma (;).

Volvamos ahora sobre la cuestión de utilizar una variable para leer un registro. Los registros de los ficheros secuenciales van separados entre sí por el código 13 (el código de ENTER) que equivale a una nueva línea. Cuando se ejecuta una instrucción PRINT (sin punto y coma al final) envía automáticamente este código para empezar una nueva línea sea cual sea el número de resultados impresos. Lo mismo ocurre en un fichero. A su vez, la instrucción INPUT del ZX-SPECTRUM requiere un ENTER para cada variable de la lista. En consecuencia, cuando esta instrucción lea de un fichero, solamente encuentra un ENTER por cada registro (por cada línea) y considera a toda la línea como un campo y, por tanto, hay que grabarlos en registros separados. Esto se puede lograr utilizando varias instrucciones PRINT o mediante un truco que ahora explicaremos. En el ZX-SPECTRUM, además de los separadores estándar que son la coma (,) y el punto y coma (;) existe un tercer separador, el apóstrofe. Para ver cómo funciona, escriba el siguiente programa:

```
NEW
10 REM USO GENERAL
20   PRINT "A=";2+2
30   PRINT "A=",2+2
40   PRINT "A="'2+2
```

Las dos primeras líneas dan el resultado que ya conocemos. La tercera línea escribe los resultados en dos líneas distintas. Por tanto, el

apóstrofe ('), realiza un salto de línea y equivale a un ENTER, o dicho de otro modo, equivale a la utilización de dos instrucciones PRINT. Operando de esta forma, construyendo registros de un solo campo, podremos leer por separado cada uno de los datos.

17.1.6.3 Cierre

Cuando se han terminado las operaciones con un fichero hay que cerrarlo. La instrucción CLOSE realiza esta tarea. Su sintaxis es

CLOSE número

Esta instrucción se encuentra sobre la tecla del 5 e incluye el símbolo de sostenido (). Al cerrar un fichero abierto por OPEN, se desconecta el canal asociado y por tanto se impide cualquier manipulación sobre los datos del fichero. Por su parte, el canal desconectado queda libre y se puede asignar a otro fichero o dispositivo.

El cierre de un fichero al terminar su utilización es especialmente importante en el caso de la grabación. El ZX-SPECTRUM utiliza un «buffer» o memoria intermedia de 512 byes y solamente transfiere los datos al microdrive cuando este buffer está lleno. Por tanto, es muy probable que al terminar el programa queden todavía algunos datos en esta memoria sin transferir. La instrucción CLOSE antes de desconectar el canal, se encarga de enviar estos datos al fichero magnético, evitando cualquier pérdida.

Por otra parte, en los ficheros secuenciales (que son los utilizados por el microdrive), la instrucción CLOSE realiza además otra operación. Esta operación consiste en colocar la marca de fin de fichero (o EOF por sus siglas de End, of File en inglés). Si se deja un fichero sin cerrar, no podrá ser utilizado posteriormente, aunque sí puede borrarse.

Por último, tenga en cuenta que los canales 0, 1, 2 y 3 no pueden cerrarse ya que están reservados para el BASIC.

17.1.7 Extensión de un fichero

El BASIC del ZX-SPECTRUM decide por su cuenta cuándo un fichero se abre para lectura o para escritura. Recordemos que un fichero secuencial no puede estar abierto simultáneamente para lectura y escritura. Si al efectuar un OPEN el fichero no existe, se crea y se abre para escritura o grabación. En caso contrario, se abre para lectura. La pregunta surge inmediatamente; ¿cómo hacer para añadir información a un fichero existente? La solución está en crear un fichero nuevo donde se transferirán los datos antiguos y los nuevos. Por ejemplo, veamos cómo se haría para añadir una nueva dirección al fichero DIRECCIÓN creado anteriormente. El programa es el siguiente:

```
10 REM REQUIERE MICRODRIVE
20 REM EXTENSION DE UN ARCHIVO
30   OPEN #4;"m";1;"DIRECCION"
40   OPEN #5;"m";1;"DIRECCION2"
50   FOR I=1 TO 4
60        INPUT #4;A$
70        PRINT #5;A$
80        NEXT I
90   INPUT "NOMBRE:";N$
100  INPUT "DIRECCION:";D$
110  PRINT #5;N$,D$
120  CLOSE #4 : CLOSE #5
```

En la línea 30 se abre el fichero DIRECCIÓN para lectura y en la 40 el fichero DIRECCIÓN2 se abre para grabación.

El bucle que va desde la 50 a la 80 transfiere la información del primer fichero (a través de canal 4) al segundo (a través del canal 5). A continuación, el programa pide los nuevos datos y los añade al segundo fichero. Finalmente, en la línea 120 se cierran ambos ficheros.

El archivo antiguo (DIRECCIÓN) puede guardarse como copia de seguridad o bien puede borrarse.

17.4 ◄

Capítulo 18

ESQUEMA DE CONTENIDO

Iniciación al lenguaje máquina.	Función PEEK.	
	Función POKE.	Ejemplo 1
		Ejemplo 2
		Ejemplo 3
	Función USR	
	Función CLEAR	
	Instrucciones IN y OUT.	
Sistema operativo.	Inicialización del cartucho.	
	Listado del directorio.	
	Borrado de ficheros.	
	Copia de ficheros.	
	Protección de ficheros.	

18.1 INICIACIÓN AL LENGUAJE DE LA MÁQUINA

Estudiaremos a continuación una serie de instrucciones para acceder al núcleo del ordenador y realizar algunas operaciones especiales. Queremos advertirle que estas instrucciones están fuera del comportamiento estándar del BASIC. Todo lo que se diga en este apartado sirve única y exclusivamente para el ordenador ZX-SPECTRUM. Los programas no son adaptables fácilmente a otros modelos, sino que generalmente hay que cambiarlos radicalmente.

Será conveniente antes de seguir, repasar la estructura de la memoria del ZX-SPECTRUM explicada en el capítulo 8 del segundo tomo.

18.1.1 Función PEEK

Con esta función podemos consultar el contenido de una posición (de un byte) de la memoria. La sintaxis es idéntica a las demás funciones, es decir

PEEK(X)

La palabra PEEK se encuentra sobre la letra O. En el ZX-SPECTRUM se pueden suprimir los paréntesis. Si lo hace, ponga mucha atención en el orden de prioridad de los operadores.

Para ver cómo funciona, escriba el siguiente programa que imprime el contenido de las 10 primeras posiciones de memoria del ordenador. Estas posiciones corresponden a la memoria de tipo ROM, y por tanto se pueden consultar pero no alterar.

```
10 REM Consultar memoria ROM.
20 FOR I = 1 TO 10
30      PRINT I,PEEK(I)
40      NEXT I
```

Probablemente le parecerá que la lista de números que aparecen no tiene sentido. Sin embargo la CPU sí que sabe interpretarlos ya que se trata de código máquina. No olvide que la función PEEK nos da la representación en decimal del contenido binario.

Una aplicación interesante de este programa nos servirá para realizar un volcado en pantalla del contenido de una zona determinada de memoria. Puesto que en memoria se almacenan instrucciones y datos, para facilitar la comprensión haremos lo siguiente. Si la posición tiene significado de carácter, se escribirá como tal. En caso contrario se escribirá un punto. El listado es el siguiente:

```
10 REM Volcado de memoria
20     INPUT "INICIO:";P1
30     INPUT "FINAL:";P2
40     FOR I = P1 TO P2
50         LET A = PEEK(I) : LET A$ = "."
60         IF 31<A AND A <128 THEN LET A$=CHR$(A)
70         PRINT A$
80         NEXT I
```

Al ejecutar el programa, éste le pedirá la posición inicial y final de la zona de memoria a explorar. Pruebe, por ejemplo, la zona a partir de 5012 como inicio y el 5459, que es donde residen los mensajes. Otra prueba interesante es la siguiente. Escriba sin borrar el programa

```
PRINT PEEK(23635)+256*PEEK(23636)
```

El valor que obtenga lo utiliza como valor inicial de la zona de memoria. Esta posición que ha obtenido es precisamente donde está almacenado su programa. Esperamos que no haya omitido la instrucción REM.

18.1.2 Instrucción POKE

Esta instrucción nos permite alterar el contenido de una posición de memoria. La sintaxis general coincide con la estándar, es decir:

POKE dirección, contenido

La instrucción POKE se encuentra sobre la tecla O y hay que manejarla con mucho cuidado. Usándola incorrectamente es muy fácil bloquear el ordenador y perder toda la información almacenada. Para comprobarlo pruebe la siguiente instrucción (asegúrese de que el ordenador no contiene información importante):

```
10 POKE 23635,0
20 POKE 23636,0
```

Después de escribir RUN intente listar el programa. Verá que lo ha eliminado.

Mediante la instrucción POKE se pueden obtener algunos efectos especiales en gráficos, imposibles de conseguir con las instrucciones normales.

Para iniciarnos en este terreno, escribamos el siguiente programa que dibuja posiciones coloreadas al azar:

```
10 REM Posiciones coloreadas
20     LET P=22527
30     FOR I = 1 TO 100
40         POKE P+RND*700,RND*127
50         NEXT I
```

Como ya conoce, la información de la pantalla está almacenada en la memoria del ordenador. La zona donde se almacenan los atributos de un carácter tienen precisamente la dirección 22528.

18.1.2.1 Ejemplo 1

Una utilidad importante de la instrucción POKE es la definición de caracteres gráficos. El ZX-SPECTRUM dispone de una serie de caracteres gráficos que se pueden obtener pasando el teclado a modo G o bien empleando la función CHR$. Aparte de estos caracteres, el usuario puede definir su propio conjunto de símbolos gráficos. Para estos símbolos se emplean los códigos comprendidos entre el 144 y el 164. En estas posiciones, el ZX-SPECTRUM almacena inicialmente letras minúsculas. Puesto que cada símbolo ocupa un cuadrado de 8 pixels de lado, se necesitarán 8 bytes para definirlo. El problema está en conocer la dirección de memoria donde se almacena cada símbolo. Para ayudarnos en este punto, el ZX-SPECTRUM dispone de una variante de la función USR. Esta función se estudiará más profundamente en otro apartado de este capítulo. Veamos a continuación un programa que nos servirá para definir símbolos gráficos. La función USR se encuentra en la tecla L.

```
 10 REM Definición de carácter gráfico
 20     INPUT "LETRA:";L$
 30     FOR I = 0 TO 7
 40         READ F
 50         POKE USR(L$)+I,F
 60         NEXT I
 70    DATA BIN 00011000
 80    DATA BIN 00100100
 90    DATA BIN 01000010
100    DATA BIN 10000001
110    DATA BIN 10000001
120    DATA BIN 01000010
130    DATA BIN 00100100
140    DATA BIN 00011000
```

Antes de seguir conviene remarcar algunos puntos de este programa. La función USR de la línea 50 nos devuelve la dirección donde se almacena el primer byte del carácter contenido en L$. Puesto que la variable I tiene valores crecientes, el programa irá variando el siguiente byte a cada vuelta del bucle.

La variable F tiene el valor de un byte del símbolo obtenido de las instrucciones DATA del final. Para mayor claridad, los valores se han representado en binario. De esta forma se aprecia claramente que se trata de un rombo. En la figura 1 se muestra la estructura de los pixels equivalente a estos datos.

Observe que al lado de cada fila de la figura se han indicado la línea que la dibuja. En los cuadrados que están en blanco hay cero y en los cuadrados marcados en negro hay un 1. Ahora, si se fija en las líneas DATA del programa verá que los unos ya forman un rombo. Sobre una cuadrícula de 64 cuadros (8 x 8) puede hacer usted otro dibujo distinto, por ejemplo una cruz. Solo tenga cuidado de que los cuadros negros sean unos y los blancos ceros.

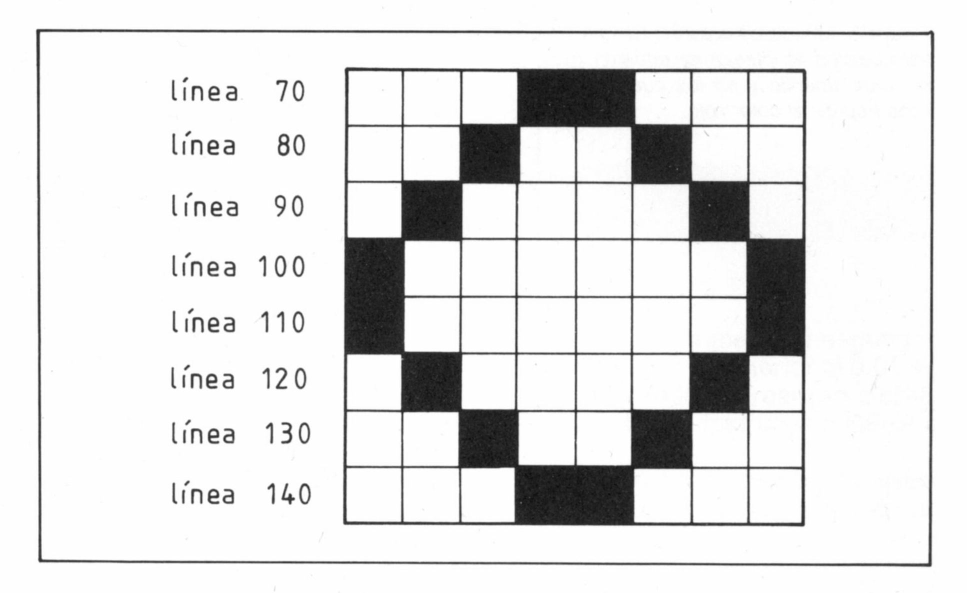


Figura 1. Definición de un carácter gráfico.

Ejecute ahora el programa. A la pregunta de la línea 20 responda, con una letra minúscula. Una *a*, por ejemplo. Entonces, el programa establece la definición de este carácter gráfico. Para visualizarlo, escriba

```
PRINT CHR$(144)
```

Esta instrucción escribirá un rombo en pantalla. Otro modo de utilizarlo es, cuando pida la letra, pasar el teclado a modo G (gráfico) y pulsar la tecla A. Veremos que en el lugar de la entrada de datos aparece un rombo.

18.1.2.2 Ejemplo 2

Como ya sabemos, el ZX dispone de una gama de 8 colores básicos. Aprovechando las posibilidades de definición de caracteres gráficos podemos obtener colores no estándar. La forma de lograrlo consistirá en colocar, lo más mezclados posible, dos colores distintos de modo que mirando desde lejos aparezcan con un color único. El programa para hacerlo es el siguiente

```
10 REM Colores no estándar
20 FOR I = 0 TO 6 STEP 2
30     POKE USR("a")+I,85
40     POKE USR("a")+I+1,170
50     NEXT I
60 INK 2:PAPER 6: PRINT CHR$(144)
```

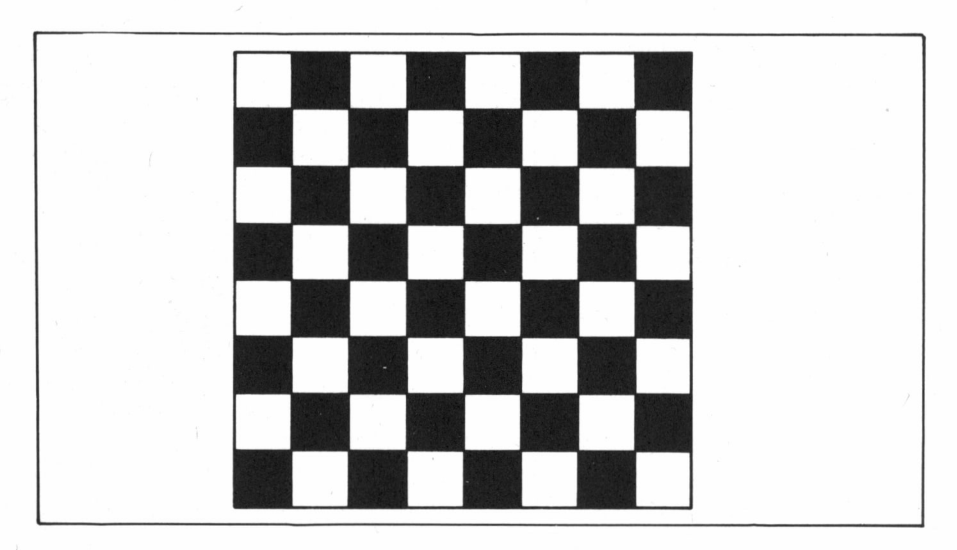


Figura 2. Mezcla de colores. En los cuadros en blanco se sitúa el color amarillo y en los cuadros negros el color rojo.

En las líneas de la 20 a la 50 se construye el carácter gráfico. Los valores 85 y 170 valen en binario (repase el capítulo 3 del primer volumen).

85 = 01010101

170 = 10101010

Con ellos se construye un carácter gráfico como el representado en al figura 2.

En la línea siguiente, se coloca la tinta roja y el papel amarillo. Al escribir el carácter 144 se obtendrá un color anaranjado. Para apreciar mejor el efecto rellene una zona de pantalla con este color añadiendo las siguientes instrucciones al programa.

```
70 FOR I = 1 TO 64
80    PRINT CHR$(144);
90    NEXT I
```

Pruebe otras combinaciones de colores (línea 60) para observar el efecto resultante.

18.1.2.3 Ejemplo 3

Dentro de la zona de memoria destinada al sistema operativo del ordenador existen una serie de posiciones especiales que contienen información destinada a controlar el funcionamiento de la máquina. Algunas de ellas no se pueden alterar ya que comprometerían el buen funcionamiento del ordenador. Sin embargo, hay otras que son inofensivas y sólo afectan a cuestiones de detalle. Veremos a continuación una serie de casos prácticos.

A lo largo de la Enciclopedia hemos visto programas para imitar el funcionamiento de un reloj. En todos ellos teníamos el problema de la sincronización en un reloj real. De hecho esta sincronización sólo se puede conseguir con cierta exactitud si el programa puede acceder al reloj interno del ordenador. El ZX guarda la información del reloj en las

posiciones 23672, 23673 y 23674. En estas posiciones se cuenta el tiempo transcurrido desde la puesta en marcha del ordenador. El programa siguiente nos muestra cómo utilizarlo.

```
10 LET T = ( 65536*PEEK(23764)+256*PEEK(23673)+PEEK(23672) )/50
20 PRINT AT 1,1;T;" SEG."
30 GO TO 10
```

Al ejecutarlo en pantalla se irá visualizando el número de segundos transcurridos. Este reloj es bastante exacto (el error es inferior al 0.01 %) pero tiene la desventaja de que se detiene cuando se realizan ciertas operaciones como lectura/grabación en cassette, la instrucción BEEP o cuando se manejan periféricos.

Alterando el contenido de ciertas posiciones podemos obtener efectos curiosos. Por ejemplo, la posición 23692 que controla el «scroll» de pantalla. Escriba el programa

```
10 FOR I = 1 TO 500
20    PRINT I,I*I
30    NEXT I
```

Como sabemos, cada vez que se llene la pantalla el ordenador nos hará la pregunta «scroll?». Para inhibir este efecto incluiremos la instrucción.

```
5 POKE 23692,255
```

Al volver a ejecutar el programa, el listado proseguirá hasta que I valga 500. Otras posiciones interesantes son las que controlan el teclado. Por ejemplo pruebe en modo inmediato

```
POKE 23609,255
```

Cuando pulse nuevas teclas observará que el «click» se ha convertido en un sonido más intenso. Puede probar con otros valores distintos de 255. Si se utiliza en el sonido se convierte en imperceptible. Más útil es el control de repetición del teclado. Para ello se disponen de dos posiciones. La primera es la 23561 que controla el tiempo que debe mantenerse pulsada la tecla antes de empezar la repetición. La segunda es la 23562 que contiene el tiempo de retardo entre repeticiones sucesivas. Estos tiempos se miden en 1/50 de segundo. En condiciones normales valen 35 y 5 respectivamente, es decir

$$35 \times 1/50 = 0.7 \text{ segundos}$$
$$5 \times 1/50 = 0.1 \text{ segundo}$$

El siguiente programa aumenta la velocidad de repetición (útil para producir líneas de guiones, por ejemplo), y a la vez incrementa el tiempo de espera antes de empezar la repetición.

```
10 POKE 23561,50
20 POKE 23562,2
```

Teclee RUN y luego para probar su efecto escriba

```
30 PRINT "..............."
```

Observará que al mantener pulsada la tecla del guión (–), la repetición tarda un poco más en producirse pero luego al ejecutar el programa el guión se escribe a doble velocidad

18.1.3 Función USR:

La función URS está destinada a que el usuario (USR proviene del inglés «user», usuario en castellano) controle ciertas posiciones especiales de la memoria. Su misión fundamental es convertir su argumento en una dirección de memoria. El argumento puede estar expresado de varias formas. Anteriormente, al definir caracteres especiales hemos visto un ejemplo de utilización. Fijémosnos que la expresión

USR («a») + I

significa la dirección de memoria donde se almacena la descripción gráfica del carácter «a», más el valor de la variable I. Otras veces el argumento es directamente numérico. Entonces, por ejemplo

USR (22500)

significa la dirección de memoria 22500. Al tratarse de una función especial no se cumple la norma de homogeneidad en los argumentos.

La función USR se emplea fundamentalmente para dos objetivos. Ya hemos visto que utilizada conjuntamente con POKE sirve para acceder cómodamente a ciertas posiciones especiales. El segundo objetivo es ejecutar subrutinas en código máquina.

Para introducir una rutina en código máquina se emplea la función USR en combinación con POKE. Para ejecutar la rutina se utiliza directamente la función, por ejemplo, dentro de LET o PRINT. La instrucción

```
PRINT USR(32000)
```

pasa control a la rutina que empieza en la posición 32000. Podríamos preguntar qué ocurrirá en este caso con el resultado de la función o, en

otras palabras, qué es lo que escribirá la instrucción PRINT. La respuesta es que al ejecutar un subprograma en código máquina, la función USR devuelve el contenido del registro combinado BC. Como sabemos, el microprocesador Z80 tiene varios registros de 8 bits algunos de los cuales se pueden combinar formando registro de 16 bits.

Veamos a continuación un pequeño ejemplo de cómo se realiza esta operación. El programa más sencillo en código máquina para que nos dé un valor en el registro BC consiste precisamente en cargar dicho registro con un valor y retornar al programa principal. El programa sería (no lo escriba)

```
LD    BC,35
RET
```

Este programa lo cargaremos en la dirección 32500. Su equivalente numérico es

DIRECCIÓN	dec	Hex
32500	1	01
32501	35	23
32502	0	00
32503	201	C9

El primer valor (el 1) corresponde a LD BC. El 35 y el 0 corresponden a los dos bytes que hay que llenar en BC. Finalmente el 201 corresponde a RET. Escriba el siguiente programa

```
10 LET d= 32500 : LET a=d
20 READ n
30 IF n<0 THEN GO TO 70
40    POKE a,n
50    LET a = a+1
60    GO TO 20
70 PRINT USR(d)
80 DATA 1,35,0,201,-1
```

La primera parte del programa, hasta la línea 60, carga las instrucciones en la dirección 32500 y siguientes. El bucle de las líneas 20 a 60 se detiene cuando encuentra un valor negativo que indica el final. Si ejecuta este programa, observará que el PRINT de la línea 70 escribe un 35.

Veamos a continuación otro ejemplo. No obstante, un programa que realice algo de cierta utilidad, escrito en código máquina adquiere una longitud desmesurada. Esta es la razón por la cual se diseñaron los lenguajes de alto nivel como el BASIC. A causa de esta dificultad, construiremos un pequeño programa para manejar los operadores lógicos vistos en el capítulo 6 del segundo tomo. Como ya sabe, estos operadores no actúan de forma estándar en el ZX. Sin embargo, el programa en código máquina que vamos a construir sí lo hará.

Empezaremos con una versión sencilla. El programa lo almacenaremos en la dirección 32500

DIRECCIÓN			HEX	DEC
32500	LD	A,25	3E	62
			19	25
32502	LD	B,30	06	6
			1E	30
32504	AND	A,B	A0	160
32505	LD	C,A	4F	79
32506	LD	B,0	06	6
			00	0
32508	RET		C9	201

Este programa realiza la operación AND entre los números 25 y 30 que, como sabemos, da 24. En primer lugar carga estos valores en los registros A y B. Luego realiza la operación en cero, de modo que el registro combinado BC contenga únicamente el resultado. Esta primera versión del programa coincide con el anterior cambiando únicamente el DATA por

```
80 DATA 62,25,6,30,160,79,6,0,201,-1
```

Al ejecutarlo comprobará que el resultado es 24.

No debemos creer que un programa en código máquina es estático y siempre actúa sobre los mismos datos. Vamos a mejorar el programa anterior para que nos pregunte los números sobre los que queremos que actúe y para que podamos elegir el operador.

El listado es el siguiente

```
 10 LET d=32500
 20 POKE d,62    :POKE d+2,6
 30 POKE d+5,79  :POKE d+6,6
 40 POKE d+7,0   :POKE d+8,201
 50 INPUT "Valor 1 :";v1
 60 INPUT "Valor 2 :";v2
 70 INPUT "and,or,xor"; a$
 80 IF a$="and" THEN LET op=160 : GO TO 120
 90 IF a$="or"  THEN LET op=176 : GO TO 120
100 IF a$="xor" THEN LET op=168 : GO TO 120
110 GO TO 70
120 POKE d+1,v1 :POKE d+3,v2
130 POKE d+4,op
140 PRINT v1;" ";a$;" ";v2;"=";USR(d)
150 GO TO 50
```

En la primera parte del programa, hasta la línea 40, se introduce en la memoria la parte fija del código. A continuación se piden los datos que

faltan. En las líneas 80 a 100 se transforma el operador escrito en letras a su correspondiente código máquina. Por ejemplo, la palabra «AND» se convierte en 160. Estos valores se introducen en las posiciones de memoria adecuadas y finalmente se ejecuta el programa en código máquina (función USR). Con este programa podemos practicar con los operadores del capítulo 6. Concretamente pruebe los ejercicios de autocomprobación 46 a 50. Al entrarle los operadores en las líneas 70, 80, 90 y 100 debe hacerlo con todas las letras y no con la tecla correspondiente.

18.1.4 Instrucción CLEAR

Ahora que estamos profundizando en el interior de la máquina, es el momento de completar el estudio de la instrucción CLEAR iniciado en el capítulo 8 del segundo tomo.

Como ya vimos, la memoria del ordenador se divide en varias zonas. Dentro de la zona destinada al usuario tenemos el área del programa, el área de las variables, el área dinámica y la zona destinada a los gráficos del usuario. El área dinámica tiene la particularidad de que su tamaño crece o decrece según las necesidades del programa que se ejecuta. Obviamente, el crecimiento de este área está limitado a un cierto valor, pues de lo contrario invadiría la zona de gráficos destruyéndolos. Este límite se denomina RAMTOP.

La instrucción NEW (y también RUN) borran el contenido de la zona de variables y del área dinámica hasta la posición señalada por RAMTOP. Dejando intacta la zona destinada a gráficos. Mediante la instrucción CLEAR seguida de un número podemos cambiar el límite del área dinámica, de modo que tengamos más espacio para gráficos y menos para programa o vice-versa.

El primer paso es averiguar cuanto vale RAMTOP. Su valor se encuentra almacenado en las posiciones 23730 y 23731. Para averiguar su valor escriba

```
PRINT PEEK(23730)+256*PEEK(23731)
```

Para cambiar su valor, simplemente escriba CLEAR seguida de un número. Por ejemplo:

```
CLEAR 3000
```

Esta instrucción produce los siguientes efectos:

- Borrado de todas las variables.
- Borrado de plantalla (CLS).
- La posición inicial de PLOT pasa a ser (0,0).
- Ejecuta un RESTORE.
- Observa si puede cambiar el valor de RAMTOP. Si es así, el nuevo valor será el indicado en la instrucción. En caso contrario lo deja invariable.

Como ejemplo, vamos a aumentar desmesuradamente el tamaño de la zona de gráficos de modo que no quede apenas espacio para el programa.

De este modo observaremos qué ocurre cuando se llena la memoria. Escriba:

```
NEW
CLEAR 23800
```

Le dará el mensaje de «RAMTOP no good», es decir, que es incorrecto. Ahora cambie y escriba:

```
NEW
CLEAR 23900
```

A continuación comience a escribir un programa y verá que enseguida la máquina se le bloquea, pues no hay sitio para lo que está introduciendo.

La utilización de la instrucción CLEAR está limitada para aquellos casos en que el programa requiere un gasto muy elevado de memoria, ya sea para datos o para gráficos. Estos casos son poco frecuentes, pero si se encuentra en uno de ellos, probablemente necesitará saber calcular cuánta memoria gasta un programa. A continuación le daremos las normas para realizar este cálculo.

El gasto total de memoria será igual a la suma de la memoria ocupada por el programa más las variables. Para cada línea de programa se aplica las siguientes reglas:

a) El número de línea ocupa 2 bytes.

b) Cada carácter de la línea ocupa 1 byte. Tenga en cuenta que las palabras reservadas del BASIC ocupan 1 sólo carácter.

c) Las constantes numéricas ocupan 6 bytes. El primero de ellos es un código identificativo y los 5 restantes almacenan el código ENTER.

Veamos un ejemplo. La línea

```
50 PRINT "Total=";5*A
```

ocupará la siguiente cantidad de memoria:

Número de línea (50)	2
Palabra PRINT	1
Texto «TOTAL=»	8
Punto y coma (;)	1
Número (5)	6
Signo multiplicar (*)	1
Letra A	1
Código ENTER	1
	21 bytes

Obviamente, la ocupación total de un programa equivaldrá a la suma de las ocupaciones de cada línea. Este gasto de memoria es la ocupación estática, es decir, antes de ejecutar el programa. Todavía no se ha reservado espacio para las variables y los conjuntos dimensionados. Para calcular la memoria necesaria para las variables se aplicarán las siguientes reglas:

- Para las variables numéricas se reservan tanto bytes como letras tenga el nombre de la variable más 5 bytes para el número.
- Las variables que controlan un bucle FOR/NEXT ocupan 19 bytes que corresponden al nombre de la variable (1 byte), el valor actual (5 bytes), al límite (5), al incremento (5), el número de línea actual dentro del bucle (2) y al número de instrucción dentro de dicha línea (1).
- Las variables textuales ocupan un byte para el nombre, 2 bytes para la longitud y a continuación, tantos bytes como caracteres tenga el texto almacenado, que puede ser cero.
- Para los conjuntos dimensionados se emplea 1 bytes para el nombre, 2 bytes para almacenar el número total de elementos, 1 byte para especificar el número de dimensiones, y luego 2 bytes para cada dimensión. A continuación se sitúan los elementos. Si el conjunto es numérico se emplean 5 bytes por cada elemento. Si es textual, se emplea 1 byte para cada uno.

Veamos unos cuantos ejemplos:

Las variables de la línea:

```
10 LET total = 24+b : LET a$="CASA"
```

ocupan

TOTAL	tendrá 5	bytes	del	nombre	+ 5	bytes	del contenido	= 10 bytes
B	" 1	"	"	"	+ 5	"	" "	= 6 bytes
A$	" 1	"	"	"	+ 2	"	longitud + 4 del texto	= 7 bytes

Los conjuntos dimensionados de la siguiente línea ocuparán

```
10 DIM a(20): DIM b$(8,12)
```

A:		B$:	
Nombre	1	Nombre	1
Número elementos	2	Número elementos	2
Número dimensiones	1	Número dimensiones	1
Primera dimensión	2	Primera dimensión	2
Elementos (20 × 5)	100	Segunda dimensión	2
	106	Elementos (8 × 12 × 1)	96
			104

Aparte de la memoria de programa y de datos tenemos el área dinámica que suele ser pequeña comparada con las otras dos. Más importancia puede tener la memoria destinada a establecer los canales de entrada/salida, por ejemplo a impresora o microdrive. En el SPECTRUM, estos canales ocupan parte de la memoria del usuario. En el caso del microdrive, este gasto es de 595 bytes y, en consecuencia, es importante tenerlo en cuenta.

18.1.5 Instrucciones IN y OUT

Realmente, estas dos instrucciones carecen por completo de interés para los usuarios normales de un ordenador. Sin embargo, pueden ser de gran utilidad para aquellos aficionados a la Electrónica que deseen conectar aparatos o dispositivos al SPECTRUM.

Estas dos instrucciones de BASIC se corresponden directamente con dos instrucciones de código máquina del microprocesador Z80 con el mismo nombre. Su función es leer o enviar información a los circuitos de entrada/salida que en lenguaje técnico se denominan «ports». Internamente el ordenador hace un uso intensivo de estas dos instrucciones para comunicarse con el teclado, con la pantalla, la impresora, etc.

La función IN obtiene un byte de un «port» determinado. Se encuentra en la tecla R. Es análoga a la función PEEK, sólo que opera con los «ports» en lugar de la memoria. La sintaxis es

IN (d)

en donde *d* es la dirección del «port» a consultar.

La instrucción OUT envía un byte a un «port». Es análoga a la función POKE. Se encuentra en la tecla O. La sintaxis es

OUT d,v

en donde *d* es la dirección y *v* es el valor a enviar.

Los buses de direcciones, de datos y de control son accesibles en la parte posterior del Spectrum, en el conector de impresora. Por lo tanto, puede conectar ahí un aparato y gobernarlo con un programa en BASIC. Como esta tarea está reservada a expertos en electrónica, daremos un ejemplo de utilización de IN y OUT con un dispositivo preexistente: el teclado.

El teclado está dividido en 4 filas y cada una de ellas en 2 semifilas con 5 teclas. Cada semifila está conectada a un port diferente. Las direcciones de estos ports son

SEMIFILA	TECLAS		DIRECCIÓN
0	CAPS SHIFT	a V	65278
1	A	a G	65022
2	Q	a T	64510
3	1	a 5	63486
4	0	a 6	61438
5	P	a 7	57342
6	ENTER	a H	49150
7	SPACE	a B	32766

En el ZX-SPECTRUM + las direcciones son los mismos. No tenga en cuenta la fila de la barra espaciadora. La dirección de cada semifila puede calcularse con la fórmula 254 + 256 * (255-2↑n) en donde n es el número de semifila. El siguiente programa nos permitirá comprobar el funcionamiento de IN.

```
10 REM Consulta del teclado
20 INPUT "Semifila:";n
30 LET d = 254+256*(255-2^n)
40 LET t = IN(d)
50 PRINT AT 0,0;"Tecla=";t;
60 GO TO 40
```

En primer lugar el programa pide el valor de la semifila para lo cual contestaremos un número entre 0 y 7 según la tabla anterior. Seguidamente calcula la dirección del «port» y lo utiliza en la función IN para obtener el número de la tecla. Finalmente, lo escribe en pantalla y consulta de nuevo el teclado. Para detener el programa pulse la tecla BREAK. La función IN devuelve un byte que en el caso del teclado tiene el siguiente significado. Los primeros 5 bits corresponden a cada una de las teclas. El bit 0 corresponde a la más exterior y el bit 4 a la más interior. El bit 6 corresponde a la conexión EAR y el 7 no es utilizado. Cuando la tecla está pulsada, el bit asociado vale 0 y 1 cuando no lo está.

Observará que al ejecutar el programa, la función IN detecta cuándo se pulsa y cuándo se suelta la tecla. En este punto difiere del comportamiento de INKEY$ que sólo detecta las pulsaciones.

18.2 SISTEMA OPERATIVO

El ZX tiene sistema operativo que constituye un ejemplo típico de sistema operativo sencillo incluido dentro del propio lenguaje BASIC. Por consiguiente, el BASIC tendrá una serie de instrucciones especiales que en este caso están destinadas fundamentalmente al manejo del microdrive. En los apartados siguientes daremos una descripción de dichas instrucciones.

18.2.1 Inicialización del cartucho

Antes de utilizar un cartucho de microdrive por primera vez hay que inicializarlo o formatearlo. Este procedimiento es típico de los dispositivos magnéticos y hay que efectuarlo también para los diskettes y los discos de ordenadores grandes. El proceso de formateado tiene como misión fundamental la división del soporte magnético en sectores físicos. Al mismo tiempo identifica aquellos que son defectuosos y les pone una marca para que no sean utilizados. Finalmente graba una etiqueta con un nombre y cierta información complementaria al principio del dispositivo. La instrucción para realizar esta operación tiene la forma general

```
FORMAT "m";n;nombre
```

en donde *n* es el número de microdrive (normalmente 1) y nombre es un texto de hasta 10 caracteres que identifica al cartucho. Por ejemplo:

```
FORMAT "m";1;"DATOS"
```

Esta instrucción se encuentra en la tecla cero y formateará el cartucho colocado en el microdrive 1 y le dará el nombre DATOS. Durante la operación, que suele tardar unos 30 segundos, observará que el contorno parpadeará, se borrará, parpadeará de nuevo y al final aparecerá el mensaje OK.

Recuerde que el formateado sólo hay que realizarlo la primera vez ya que destruye cualquier información preexistente. Esto puede ser útil en el caso que decida utilizar cartuchos que contengan información obsoleta.

18.2.2 Listado del directorio

Cuando se graban ficheros en un microdrive, el ZX mantiene simultáneamente un directorio de ficheros. Este directorio se puede consultar mediante la instrucción CAT (nombre que proviene de catálogo). Esta instrucción se encuentra sobre la tecla del 9. Para probarla, inserte un cartucho en el microdrive y escriba

CAT 1

que significa lista el directorio del microdrive 1. En la pantalla aparecerá la siguiente información:

- El nombre del cartucho asignado en la instrucción FORMAT.
- Una lista de los nombres de los ficheros.
- El espacio libre que queda en el cartucho medido en Kby.

Es conveniente utilizar la instrucción CAT inmediatamente después de formatear un cartucho. El espacio libre debe ser al menos de 85 Kby. En caso contrario, se trataría de un cartucho defectuoso.

La instrucción CAT tiene dos sintaxis posibles, que son

CAT n

CAT #m,n

La primera sintaxis es la que hemos utilizado y en ella la letra *n* representa el número de microdrive. La segunda sintaxis nos permite enviar el listado generado por la instrucción hacia un canal determinado. Por ejemplo, con la instrucción.

```
CAT #3,1
```

obtendremos el listado del directorio en la impresora. Del mismo modo

se puede enviar hacia otro fichero (abierto previamente por OPEN) de modo que un programa pueda utilizar posteriormente dicho listado.

No existe la instrucción equivalente de CAT para el cassette. Sin embargo puede utilizar un truco. Si emplea la instrucción LOAD con un nombre de programa inexistente, el ZX recorrerá toda la cinta buscándolo y mostrando en pantalla los nombres de los ficheros encontrados.

18.2.3 Borrado de ficheros

Para eliminar ficheros que ya no sean útiles se dispone de la instrucción ERASE cuya sintaxis general es

ERASE «m»;n;nombre

La palabra ERASE significa borrar y se encuentra sobre la tecla del 7. La letra *m* entre comillas es fija y especifica que se opera sobre un microdrive. La letra *n* designa el número de microdrive y finalmente el nombre es un texto o expresión textual que indica el nombre del fichero que se quiere borrar. Por ejemplo, para borrar el fichero llamado DIRECCIÓN, escribimos

ERASE «m»;1;«DIRECCIÓN»

La instrucción ERASE sirve indistintamente para ficheros que contengan programas y para ficheros que contengan datos.

18.2.4 Copia de ficheros

Una de las instrucciones más importantes de un sistema operativo es aquella que nos permite efectuar copias de ficheros dentro del mismo dispositivo o hacia otros dispositivos. Para realizar este cometido, el ZX dispone de la instrucción MOVE (mover o trasladar). La sintaxis general es

MOVE canal TO canal

La palabra MOVE se encuentra sobre la tecla del 6 y la palabra TO es la misma que la utilizada en la instrucción FOR. Según la sintaxis que acabamos de expecificar, la instrucción MOVE envía datos desde un canal hacia otro. Por ejemplo, pruebe el siguiente programa

```
10 MOVE #1 TO #2
```

(No intente escribirlo si no tiene microdrive).

Cuando escriba RUN observará que todo lo que teclee aparecerá en pantalla incluyendo la propia tecla de interrupción BREAK. Para cortar el programa la única solución consiste en pulsar varias veces la tecla ENTER hasta que aparezca la pregunta scroll? y entonces pulsa BREAK. Este programa nos ha enseñado que es peligroso manipular el canal provinente del teclado puesto que el ordenador puede quedar bloqueado.

Podríamos pensar que la instrucción MOVE necesita trabajar con ficheros abiertos para realizar la copia pero en realidad no es así.

Si en lugar de especificar un número de canal se especifica el nombre completo de un fichero, la propia instrucción MOVE se encarga de abrirlo y cerrarlo. Por ejemplo, supongamos que deseamos examinar rápidamente el contenido del fichero DIRECCIÓN. Entonces escribimos

MOVE «m»;1;«DIRECCIÓN» TO 2

Sin embargo debe tenerse en cuenta que MOVE sólo trabaja con ficheros de datos, no con ficheros que contengan programas. Si desea examinar un programa no hay más remedio que utilizar LOAD y LIST.

Si se especifican ficheros para el origen y para el destino de MOVE se obtienen copias de ficheros. Ejemplos:

MOVE «m»;1;«DIRECCIÓN» TO «m»;1;«DIRECCIÓN2»

MOVE «m»;1;«DIRECCIÓN» TO «m»;2;«DIRECCIÓN»

En el primer caso se obtiene en el microdrive 1 el fichero DIRECCIÓN2 que es idéntico al fichero DIRECCIÓN del mismo cartucho. En el segundo caso se obtiene una copia del fichero DIRECCIÓN sobre el segundo microdrive sin cambiar el nombre original.

De nuevo, recordemos que MOVE sólo trabaja con archivos de datos. Para obtener una copia de un programa debe utilizar LOAD y SAVE.

18.2.5 Protección de ficheros

El ZX permite construir ficheros cuyo nombre no aparezca en el directorio. De esta forma quedan protegidos. Para lograrlo, el primer carácter del nombre debe tener el código ASCII cero. Por ejemplo:

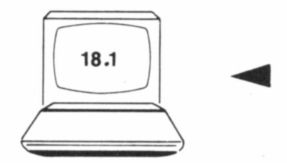
```
 10 REM Requiere Microdrive
 20 OPEN #4;"m";1;CHR$(0)+"SECRETO"
 30 FOR i= 1 TO 4
 40    READ n$,o$
 50    PRINT #4;n$,o$
 60    NEXT I
 70 CLOSE #4
 80 DATA "Juan","c/Mayor 15"
 90 DATA "Ana","Pl. Nueva 5"
100 DATA "Luis","Avda. Principal 200"
110 DATA "María","Paseo de la luna s/n"
```

Al ejecutar este programa se creará el fichero llamado SECRETO precedido de un código cero. Si ahora escribe

```
CAT 1
```

observará que dicho fichero no aparece en la lista aunque existe en realidad. El hecho de que no aparezca su nombre no impide que se pueda utilizar normalmente en una instrucción OPEN, MOVE ó ERASE. En este caso, deberá construir el nombre tal como aparece en la línea 20 del ejemplo.

Si construye un fichero protegido no olvide apuntar el nombre en alguna parte ya que si no recordase el nombre el fichero sería inaccesible.



Índice

PARTE II. PRACTICAS CON EL ORDENADOR

Capítulo 16

Capítulo 17

Capítulo 18



SPECTRUM

5

- El concepto de algoritmo
- Algoritmo de selección
- Algoritmo de burbuja
- Algoritmo de Shell
- Algoritmo Simple Merge
- Algoritmo de Zeller
- Algoritmo de búsqueda secuencial
- Algoritmo de búsqueda binaria
- Los ficheros de datos
- Ficheros secuenciales, directos e indexados
- Acceso a un fichero
- Iniciación al lenguaje máquina
- El sistema hexadecimal
- Organización interna de la CPU
- El código máquina
- Las instrucciones PEEK, POKE, CLEAR, IN y OUT
- La función USR
- Los sistemas operativos
- Lenguajes máquina y ensamblador
- Lenguajes interpretados y compilados
- Fortran, Cobol, PL/I, Algol, Pascal, C
- APL, LISP, LOGO, CL
- Procesadores de textos
- La hoja electrónica
- Las bases de datos

ceac